DESARROLLO DEL FRONT-END DE UNA APLICACIÓN WEB CON EL FRAMEWORK EXT JS 5

Autor: Enrique Carrero Salcedo
Grado: Ingeniería Telemática
Empresa con la que realiza el proyecto: Indra Sistemas Tutor en la empresa: Francisco Javier López Torralba Tutor en la universidad: Ascensión Gallardo Antolín

Título del Trabajo de Fin de Grado (TFG):

Desarrollo del Front-End en una Aplicación Web con el framework EXT JS 5

Autor:

Enrique Carrero Salcedo

Tutor en la empresa:

Francisco Javier López Torralba

Tutor en la universidad:

Ascensión Gallardo Antolín

EL TRIBUNAL

Presidente:

Guillermo Carpintero del Barrio

Vocal:

Vicente Luque Centeno

Secretario:

Alicia Rodríguez Carrión

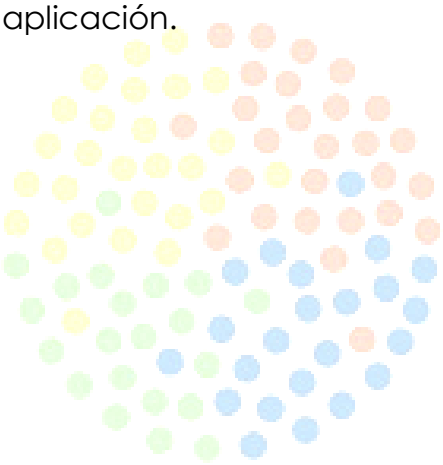
Realizado el acto de defensa y lectura del Trabajo de Fin de Grado el día 16 de octubre de 2015 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

Índice

1. RESUMEN BREVE

Este trabajo de fin de grado trata sobre el desarrollo de una aplicación web utilizando uno de los frameworks de JavaScript más potentes del mercado, el EXT JS5. Este ha sido desarrollado por Sencha.

La memoria está dividida en 9 capítulos, en los cuales se explican los conocimientos básicos que un desarrollador debe tener para comenzar a utilizar el EXT JS 5; las ventajas y los contras de este framework respecto a otros; una descripción general de cómo desarrollar el Front-End de una aplicación web con EXT JS 5 (se incluye la posibilidad de descargar una aplicación creada por el autor de la memoria para facilitar el aprendizaje del lector); se describe la aplicación que el autor ha desarrollado como proyecto fin de grado. Finalmente, se detalla el presupuesto, diagrama de Gantt y restricciones que el programador ha tenido en cuenta para el desarrollo de la aplicación.



2. INTRODUCTION (ENGLISH)

According to current systems engineers, this language has experienced an exponential growth thanks to JavaScript and his diverse libraries. This makes possible to implement a lot of programs nowadays, as for example: Arduino and Raspberry controllers, applications web, relational database and non-relational database… That is why JavaScript will be in a short time more demanded that Java itself.

In this chapter there are described basics skills that someone interested in it must have for understanding this End of Degree Project. If a programmer want use the EXT JS 5 framework, he needs to know the basic structure of a web application and JavaScript.

2.1. GLOSSARY

Acronyms used in this project report are listed below.

Expression	Description
HTML	HyperText Markup Language: Standard for making web pages. Created by W3C.
W3C	World Wide Web Consortium: International consortium that makes web recommendations.
CSS	Cascading Style Sheets. Language use for defining and creating the HTML´s appearance.
SASS	Syntactically Awesome Stylesheets: A stylesheet language. In Sass files are used scripting languages. It is necessary a compiler for translating them to Css.
HTTP	Hypertext Transfer Protocol: Protocol used in WWW´s transmission.
WWW	World Wide Web: Document distribution system what are accessible through Internet.
IP	Internet Protocol: Numerical label that identifies equipment inside a network.
URL	Uniform Resource Locator. It identifies resources according to an Internet´s standard.
URI	Uniform Resource Identifier: It identifies resources according to an Internet´s standards being unique in a network.
API	Application Programming Interface: Library with all concepts of a framework or a computer language.
DOM	Document Object Model: Standard for representing HTML and XML´s documents.
XML	eXtensible Markup Language: It defines a grammar for organizing biggest documents.
YUI	Yahoo! User Interface: JavaScript's library for creating interactive application.
RIA	Rich Internet Applications: Complex and very interactive web application.

AJAX*	Asynchronous JavaScript And Xml: Asynchronous´s request to web server that are executed in the web client. Interactive changes that take place in the web happens without the rechange of the application. This improved interactivity, speed and a better use of the application.
LGPL	Lesser General Public License: License create by the Free Software Foundationque company; it claims that a software is free for all users.
JSON	JavaScript Object Notation: Light format for the data´s exchange. Its notation is similar a JavaScript´s object.
XHTML	eXtensible Hypert Text Marku Language: Bacically HTTML documents are well-formed and followed the standard XML.
XSLT	eXtensible Stylesheet Language Transformations: W3C company´s standard that present a transformation of a XML document in other. It makes possible even to change its format.
REST	Representational State Transfer: Software architecture style for building scalable web services. It is based in the HTTP standard.
UI	User Interface: Combination of styles that are defined by programmer for a component.
JSP	JavaServer Pages:Technology that helps software developers to create dynamic web pages based on HTML and XML. It uses the Java language.
CRUD	Create Read Update Delete: Applications that use this function for getting on with the database.

[Tabla 1]

2.2. BASIC CONCEPTS

2.2.1. WEB APLICATION

In software engineering web applications are known as the programs that users can use through an internet web server or a web intranet server and a web browser. Therefore, this tool has to be programmed with languages that are supported by a browser.
The most important thing for a web application is that the web browser of every user has to operate as a web client, independently of the operating system that is executed. This generates great easily portability between the diverse platforms that can be found in the market.
Main functionality feature of a web application is to provide the user with a method of interact quicker with the database, also to send form´s requests, to take part in interactive games, to obtain information…
For designing a web application it is necessary to know HTML and CSS. With the first one, it will be designed elements that appear in the page; while with the second one, it will be designed the different styles instead; thus the application web will take the appearance that the programmer wishes.

Once that the programmer knows these two terms, he must learn to differentiate the functionality that makes the web client and the web server:

- The web server or HTTP server is a program that executes an application, making both bidirectional connections or unidirectional, and they can be synchronous or asynchronous with a web client. The web server must have to be assigned a port number. In the port all requests made by a client are listened. All web application resources share the same URL´s prefix. The URL is made of a server IP direction, where the application web is hosting, and the port number. For example: "localhost:80". This means the server is in the same machine as where the client is executing the application, and the number 80 stands for the port number where the server is listening all requests done (80 is the port number that HTTP listen automatically). Usually, the server executes and compiles the code emitted by the web browser. Apart from this it makes opportune actions for responding to the client with what server wants to show. This application part is known as Front-End. The server must make connection, must submit a query to database and must specify application web´s routes. These actions are known as the Back-End.
- The web client or web browser must show the code HTML and the CSS that receives when the web client makes requests to the web server. If the user interacts with the program interface, the browser will send new request.

Currently, there are available lots of frameworks that make easier the web application creation (as Ruby on Rails, Node JS, Angular JS, Sencha EXT JS…). These provide a lot of methods that make easier the Back-End implementation as to receive requests and to send responses, to submit a query to database, to create object through a model that represents a query to database… and Front-End implementation as elements that have already been made before by programmer´s frameworks (buttons, textfield, datefield, grid…) and different layout for organizing objects in the view. They also create models which are replenished with data that the user has inserted in the given form. The form is send to the web server. They also create stores (model´s collection) and they make request…

2.2.2. JAVASCRIPT

JavaScript is a high level, dynamic, untyped, and interpreted programming language. This language is interpreted because the code is executed without a previous compilation, and it is

untyped because variables have not got a defined type in this language. This could be an important difficulty for Java and C programmers, because variables can be "boolean", "string", "integer", "null", "undefined", "objects", "arrays", "functions"… All above depends on the data type that the variable has been associated with.

For initializing a variable, the JavaScript key "*var*" has to be placed before the variable name. If it is not done this way, the interpreter will understand that a global variable is been used. This is a big mistake because this variable can be changed in any program´s places generating an execution's error.

As an example of how to change the variable type would be:

```
var example = true; (boolean)
example = 6; (integer)
example= "This is correct in JavaScript"; (string)
example={
        name: "Enrique",
        surname{
                first: "Carrero",
                second: "Salcedo",
        },
        age: 22
}; (object)
```

Known that an object has got properties, and they have got a name and a value. If the programmer wants to access to one object´s attribute, he must do it in Java´s way. For example:

```
example.name= "Juan";
```

Even though there are no classes in JavaScript, it does exist legacy. An object can be created inherited from another. It will always have a reference from the object that inherits. For example:

```
var exampleInherited= Object.create(example);
```

The client-slide term of JavaScript makes reference to the execution´s environment of JavaScript given by browsers. This environment is made up of by a lot of APIs defined in the HTML5´s standard and other implicated. The most important JavaScript´s APIs are: jQuery and DOM.

Although this language was started to use to implement actions that were produced in the browser (as to push a button, to organize a

view, to listen an event, to change dynamically a window and so on), nowadays, all type of applications have been programmed with JavaScript.

JavaScript also implements control sentences (as if, for, switch, while…) in Java´s and C´s way.

2.3. INCENTIVES

The main reason for deciding to work in this project is to introduce a way to develop a web application with a framework non intuitive, but that it facilities the Front-End implementation.

As EXT JS uses the most popular language at the moment, a lot of software´s engineers belive that web applications will use this framework to implement the client visual part. In addition, as the most web designers know JavaScript language, they would not find any difficulties when they will work with it.

As it will be shown over this document, the learning of EXT JS 5 is complicated, but the more programmer goes forward, the less the difficulties are. This is because to know more layouts, more components or items, event´s fires…; the web designer will program faster and he will use more methods, events and configuration´s parameters. This will generate a friendly interface for the final user.

2.4. OBJETIVES

The main and principal goal of this project report is to provide basic knowledge for working with EXT JS 5. The key aspects that the programmer must bear in mind are as follows:

- All web applications introduced with this frameworks will be accessible through a browser that knows how to interpret JavaScript language, and it does not matter in which platform has been executed.
- The web application will be executed in a web server. It will receive all client´s requests.
- There is a list with all components that can be shown in a view. In order to know them, programmers must read EXT JS 5´s API. Here lied all configuration and methods that a component have.
- Learning line will be exponential.

2.5. PROJECT REPORT STRUCTURE

This document is divided into nine chapters which may show subsections. If the reader follows this project report in a structure manner, he will understand what EXT JS is about and how to program a web application with this framework:

1. **Brief summary**.
2. **Introduction**: Basic knowledge that a programmer must have for working with EXT JS. Also, there are described incentives and objectives taken into account.
3. **Sencha and EXT JS**: General description of Sencha and EXT JS.
4. **"Hola FIFA" with EXT JS 5**: Basic concepts of how to program a web application with EXT JS 5. There is an example of a program that every people can be downloaded from GitHub freely (notice that this program has been developed by the author).
5. **Complex graph interface: Monitoring of banking document´s transmissions**: Description and implementation of the End of Degree Project done at Indra.
6. **Budget, planning and restrictions**: Details about budget, planning and the restrictions involved in this End of Degree Project. Just to point out that these are not real (because is illegal) that is why the author described similar ones.
7. **Summary**: Summary of all concepts involved in this project report.
8. **Conclusion**: Conclusions and knowledge reached of how to program a web application with EXT JS 5.
9. **References**: list of all documents, pages and books that the author has searched while writing this work.

3. SENCHA Y EXT JS

Antes de empezar a usar este framework se va redactar la historia de forma breve de lo que representan Sencha y Sencha EXT JS, con lo que se pretende poner al programador en situación antes de empezar a desarrollar el Front-End de su aplicación web.

3.1. SENCHA

La Compañía de Gestión de Aplicaciones Web Sencha permite a otras empresas construir, probar, implementar y administrar aplicaciones online sin dificultades. Esta plataforma proporciona a los desarrolladores un conjunto de herramientas con múltiples funcionalidades.

Sencha ofrece distintos framework para los lenguajes de programación Java y JavaScript. Haciendo uso de HTML5, consigue aportar soluciones para equipos de sobremesa, portátiles, Smartphones y Tablets.

Sencha fue fundada en 2007, por lo que ya lleva ocho años innovando en tecnologías web, lo que significa que es un proveedor confiable de este dominio. Su sede central se encuentra en Redwood City, California. Sencha está presente en empresas de todo el mundo, por lo que cuenta con presencia local en Canadá, Reino Unido, Italia, Holanda, Hong Kong, Corea, Australia y Japón. El éxito de la compañía ha sido reconocido por los analistas del sector. La empresa de consultoría y de investigación de tecnologías de la información **Gartner** nombró a Sencha como un vendedor "visionario".

En un principio, las distintas librerías que hoy ofrece Sencha, eran distribuidas por varias empresas, las cuales tenían como base las distintas APIs de JavaScript que se pueden consultar de forma gratuita, como son *JQuery* y *YUI* (creada por Yahoo). Estas empresas se pusieron de acuerdo para fusionarse, ya que así llegarían a un mayor número de clientes. También pensaron que si creaban sus propias librerías base, los frameworks que desarrollaran con ellas, implementarían una mayor funcionalidad, pudiendo ofrecer a sus usuarios un mejor servicio técnico de resolución de problemas.

3.1.1. COMPONENTES DE LA PLATAFORMA

Con el nacimiento de la compañía, también surgieron los frameworks que hoy en día se pueden adquirir:

- Sencha EXT JS: se explicará en el siguiente sub-apartado de la memoria.

- Sencha Touch: construye aplicaciones web para dispositivos táctiles, como tabletas y móviles de última generación.
- Sencha Designer: crea impresionantes interfaces web con un simple arrastre de elementos con el ratón.
- Sencha GWT: librería de Java para la creación de RIA.

Estos frameworks también han ido evolucionando con el tiempo, ya que Sencha se ha encargado de desarrollar nuevas versiones añadiendo nuevas funcionalidades a las librerías y resolviendo las diferentes incidencias que se han encontrado. En la versión que se está desarrollando de EXT JS (la número 6), se ha decidido unir en una misma librería las funcionalidades de EXT y Touch. Ésta se encuentra disponible en forma Beta en la página oficial de Sencha.

Una de las mayores ventajas que los frameworks de Sencha ponen a la disposición de sus clientes, es la perfecta integración del método "*debugger*" en las librerías que esta empresa ha creado, ya que acelera el depurado en las aplicaciones multi-plataforma.

3.1.2. SOPORTE Y SERVICIOS

Gracias al soporte y a los servicios que ofrece Sencha, los usuarios obtienen el máximo valor de los productos que ésta presenta. La organización ofrece a sus clientes servicios profesionales de ayuda para aumentar la productividad, mejorar la calidad y reducir los costes que ocasiona la creación de aplicaciones web. También pone a la disposición de los usuarios una pequeña formación para que puedan ponerse al día rápidamente y si encuentran cualquier tipo de duda, los expertos de Sencha se la resolverán lo más rápido posible.

3.1.3. CLIENTES

En la actualidad, más de 10.000 empresas utilizan los productos de Sencha, entre las que se encuentran 60 compañías de las catalogadas **Fortune** (lista de las 100 mejores empresas para trabajar). El foro online de la compañía está compuesto por más de 2 millones de usuarios activos. También se han registrado más de 500 millones de descargas de sus productos.

3.2. SENCHA EXT JS

Sencha EXT JS (conocido antes de la formación de la compañía Sencha como EXT JS) es un framework de JavaScript especializado en diseñar las interfaces de las aplicaciones web, haciendo más flexible el manejo de los componentes de DOM y la realización de peticiones AJAX. La librería cuenta con una API fácil de consultar, en la que se

encuentran los diferentes componentes o ítems que se pueden insertar en la aplicación, los valores de configuración que podemos asignarles, los métodos que nos ofrecen, los eventos que ocasionan y los diferentes estilos que podemos aplicar mediante CSS. También se podrá consultar los distintos modos de maquetación en los cuales se presentan los ítems de la página.

3.2.1. ORÍGENES

Originalmente se construyó como una extensión a la biblioteca de YUI. Salió al mercado el 1 de abril de 2007 con la versión EXTJS 1.0. Desde la versión 1.1, EXT se podía ejecutar como una aplicación independiente. Cuando surgió la versión 3.0 se empezó a llamar Sencha EXT JS, debido a la formación de la citada compañía. En la actualidad, se puede usar como una extensión para las librerías JQuery y Prototype. A día de hoy está disponible la versión beta Sencha EXTJS 6.

3.2.2. FUNCIONALIDADES

Lo primero que se debe conocer antes de comenzar a trabajar con Sencha EXTJS es que las versiones demo están disponibles en la web oficial de Sencha. En ella se podrá consultar los términos de uso que se especifican en la licencia. Se recomienda el uso del navegador Google Chrome, debido a que la consola de JavaScript que nos proporciona es muy útil para inspeccionar el código, tanto para depurarlo como para probar los diferentes estilos que se dan a la aplicación.

Alguno de los componentes (widgets) que el framework ofrece para incluir en el Front-End de la aplicación son:

- Cuadros y áreas de texto.
- Campos para fechas.
- Campos numéricos.
- Combos.
- Radiobuttons y checkboxs.
- Elementos de datos, en los cuales se pueden ordenar las filas de forma alfabética, o reordenarlas a gusto del usuario con un simple arrastre con el ratón, ocultar columnas, etc.
- Barras de herramientas.
- Sliders.
- Ventanas.

Muchos de estos componentes pueden realizar peticiones AJAX al servidor. También contiene numerosas funcionalidades que permiten añadir funciones interactivas a las páginas HTML, como por ejemplo

quicktips para mostrar mensajes de validación e información de campos individuales.

3.2.3. NAVEGADORES COMPATIBLES

Sencha EXTJS está soportado en los navegadores web más importantes del mercado:

- FireFox 1.5+
- Safari 3+
- Chrome 3+
- Opera 9+
- Internet Explore 6+

3.2.4. VENTAJAS

Las principales ventajas de utilizar este framework son:

1. Crear aplicaciones web complejas con componentes predefinidos.
2. Evitar problemas de validación de código para que éste se ejecute bien en los diferentes navegadores.
3. El manejo de ventanas flotantes que nos ofrece es superior al ofrecido por cualquier otro framework.
4. La relación entre el Cliente-Servidor es balanceada. Esto quiere decir que el servidor puede atender a más clientes al mismo tiempo.
5. Da una mayor eficiencia en la red, ya que disminuye el tráfico debido a qué cuenta con la posibilidad de elegir que datos se desea transmitir al servidor y viceversa.
6. Comunicaciones asíncronas con el servidor. En este tipo de aplicaciones, el motor de render (se llama así al proceso de generar una imagen) puede comunicarse con el servidor sin necesidad de una acción del usuario, dándole la libertad de comunicarse con el servidor sin que el cliente se dé cuenta.

3.2.5. DESVENTAJAS

A pesar de las múltiples ventajas que ofrece, también se pueden observar algunas desventajas:

1. Necesita una plataforma, pues depende del paquete de EXTJS para obtener los resultados deseados.
2. Para algunos desarrolladores, una desventaja es que este framework no cuenta con una licencia LGPL.

4. "HOLA FIFA" CON EXT JS 5

En este capítulo se pretende crear una guía básica con la que cualquier usuario aprenda a desarrollar la interfaz gráfica de una aplicación web. Esta práctica será similar a muchos otros "hola mundo" que se implementan en los diferentes lenguajes de programación.

Debido a que la aplicación será visual, el tema elegido para ella ha sido sobre el mundo del fútbol, ya que muchas personas están familiarizadas con esta actividad.

Como este framework sólo ofrece servicios del Front-End, todos los datos que se muestren en el navegador son *mockups* de archivos JSON estáticos, que se encontrarán dentro de la propia aplicación.

Para comenzar, se explicarán los conceptos básicos que se deben tener para la implementación de la interfaz gráfica de la aplicación.

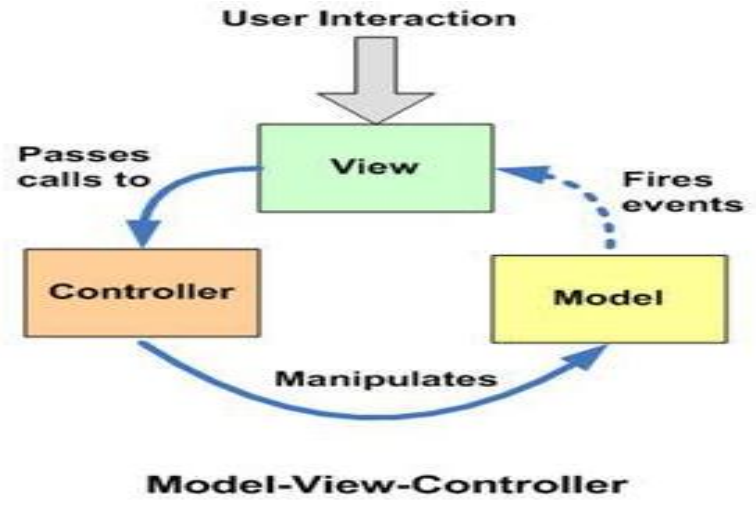
4.1. ARQUITECTURA DE EXT JS 5

La versión de este framework permite trabajar con dos arquitecturas diferentes: MVC y MVVM. Ambas comparten ciertos conceptos, ya que se fundamentan en dividir el código de la aplicación en distintos ficheros de forma lógica. A pesar de esto, los programadores de EXT JS 5 prefieren utilizar el MVVM a la hora de desarrollar sus aplicaciones.

4.1.1. MVC

Esta arquitectura representa el Modelo-Vista-Controlador (*Model-View-Controller*). La interfaz de la aplicación estará dividida en estas tres partes, lo que ayuda a organizar el código de forma lógica, haciendo que la representación de la información dependa de su funcionalidad. En la figura siguiente se muestra la forma que tienen de interactuar las distintas partes de una aplicación.

Representación del Modelo-Vista-Controlador de una aplicación.
Imagen extraída de: https://nirajrules.wordpress.com/2009/07/18/mvc-vs-mvp-vs-mvvm/

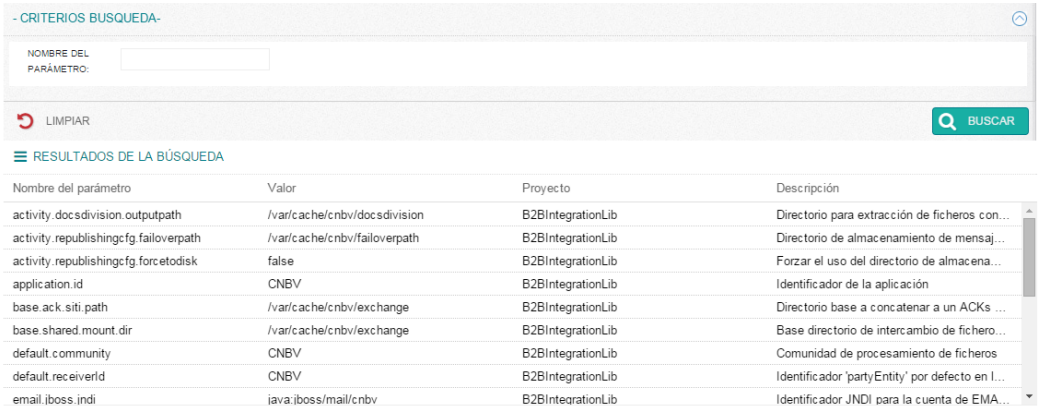


[Imagen 1]

- El modelo representa el formato común de los datos usados para la aplicación. En él se encuentran las reglas de negociación, la lógica de validación y otras funciones.
- La vista representa la parte visual de la aplicación. Es posible que se muestre la misma vista en diferentes partes de la aplicación, como por ejemplo el mismo grid o el mismo buscador.
- El controlador es la pieza central del MVC, ya que es la encargada de escuchar los diferentes eventos que ocurren en la aplicación. En él se puede manejar tanto la vista como el modelo, haciendo incluso que interactúen entre ellos.

4.1.2. EJEMPLO DE MVC

Ejemplo visual de un buscador y una lista de resultados configurados con el MVC.



[Imagen 2]

En esta sencilla captura de pantalla podemos observar la arquitectura MVC de manera muy sencilla:

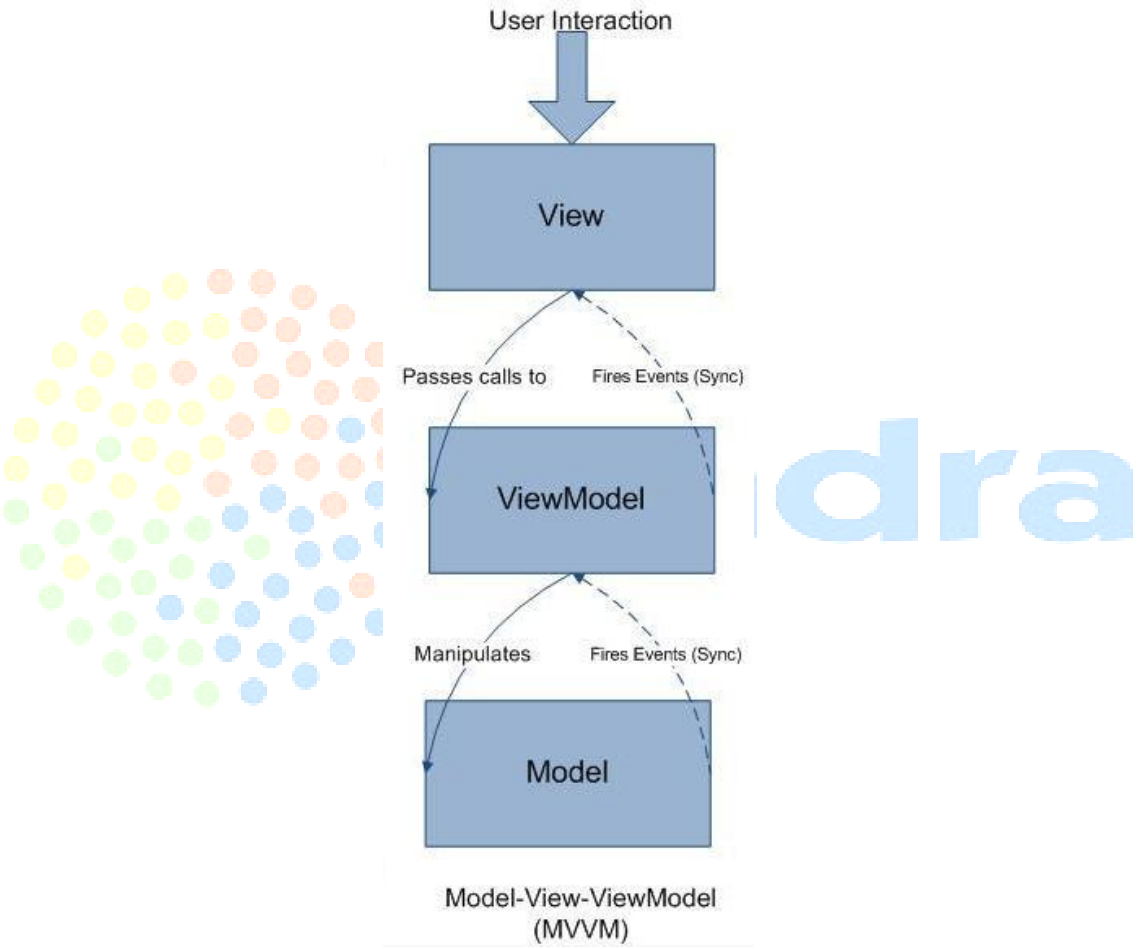
- La vista la compone un contenedor, en el que se encuentran el buscador y el resultado de la búsqueda o detalle.
- En el grid se carga el store de parámetros globales. Éste es un conjunto de modelos del tipo parámetros globales; por lo que en el ejemplo cada fila del grid representa un modelo de parámetros globales. Los atributos y validaciones que todo parámetro global debe tener están previamente definidos en su modelo.
- El controlador se encarga de obtener el dato introducido en el buscador y hacer una petición al Back-End para que éste envíe todos los parámetros globales filtrados por ese valor. Una vez recibido, se creará el store que el controlador deberá cargar en el grid. Esto se realiza cuando el usuario pulsa el botón de "Buscar". Si pulsa "Limpiar", se borrarán los datos introducidos en el campo de texto. Como se ve, el controlador interacciona tanto con la vista como con el modelo a través del store.

4.1.3. MVVM

La arquitectura Modelo-Vista-ModeloDeLaVista (*Model-View-ViewModel*) tiene como base el MVC. La principal diferencia es que el ViewModel funciona como un modelo para la propia vista. En él se

pueden añadir nuevas propiedades a la pantalla, como puede ser la de controlar los datos que se encarga de mostrar. Un ejemplo de lo que se consigue con el modelo de la vista es poder realizar "*data bindings*". Esta propiedad permite sincronizar los datos de un modelo con la vista en la que se representan. También es posible realizar una comunicación en el sentido contrario, ya que si modificas algún dato del modelo en la vista, automáticamente se cambiará en su modelo.

Representación del Modelo-Vista-ModeloDeLaVista de una aplicación. Imagen extraída de: https://nirajrules.wordpress.com/2009/07/18/mvc-vs-mvp-vs-mvvm/



[Imagen 3]

Los elementos que componen el MVVM son:

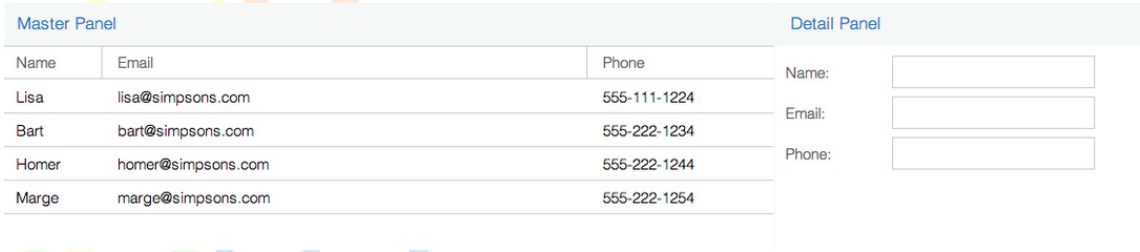
- El modelo: describe el formato de los datos comunes que deben presentar todos los elementos de este tipo dentro de la aplicación. Esto es similar al clásico MVC.
- La vista: representa los datos que el usuario observa. Esto también es similar al MVC.

- En el modelo de la vista se proporcionan los distintos destinos de enlace de datos que la vista debe representar. En muchos casos, en el modelo de la vista se expone el modelo directamente o proporciona propiedades que encapsulan partes específicas del modelo. En esta capa también se puede definir a qué datos del modelo se debe realizar un seguimiento para realizar una continua sincronización con los datos representados en la vista ("*data bindings*").

4.1.4. EJEMPLO DE MVVM

Si se vuelve al ejemplo del MVC se puede comprobar que algunos problemas que se dan entre los componentes y los modelos a los que hacen referencia se deben resolver manualmente, mientras que con el MVVM estos se pueden resolver con una simple referencia.

Ejemplo de una lista y un detalle de creación de recursos elaborado con un MVVM.



[Imagen 4]

En la figura anterior podemos ver una simple pantalla del tipo Maestro/Detalle. La vista maestro está compuesta por un grid que contiene una lista de registros. Cuando se selecciona uno de ellos, gracias a la vista modelo y al "*data bindings*" que se ha programado, los datos del panel detalle se completarán automáticamente; y si decidimos modificarlos en esta vista, también se cambiarán en el grid. Como se puede deducir, si esto se realizara con un MVC, se necesitaría navegar por las distintas pantallas para cargar el detalle y modificar el registro seleccionado, siendo un trabajo mucho más complejo.

4.1.5. CONTROLADOR EN MVVM

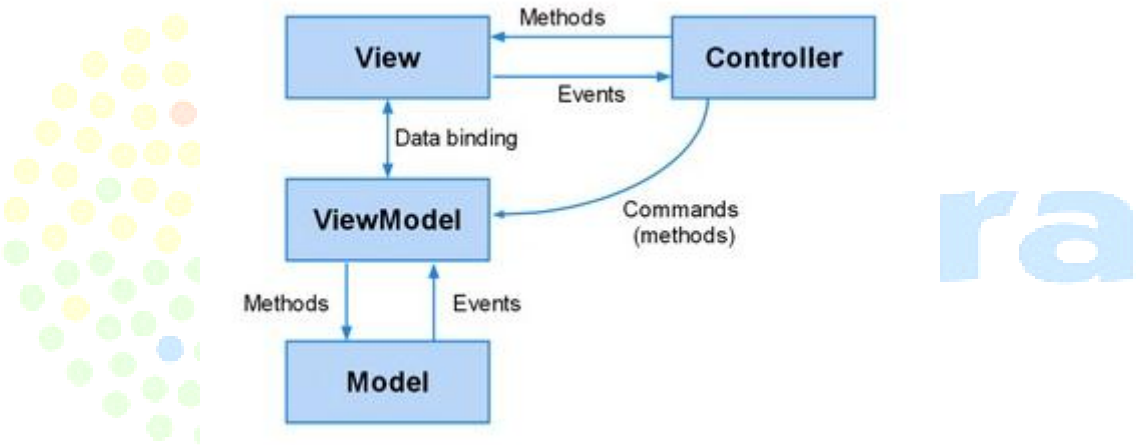
El nombre de Modelo-Vista-ModeloDeLaVista puede crear confusión, ya que muchas personas podrían pensar que en esta arquitectura no existe el concepto de controlador. Esto no puede ser más erróneo.

A pesar de que el MVVM puede soportar un controlador similar al que encontramos en la arquitectura MVC, EXT JS 5 también soporta una

nueva variante, que se denomina "controlador de la vista" (*ViewController*). Al igual que el modelo de la vista, éste nuevo controlador construye un enlace directo con la vista, lo que elimina gran parte de la sobrecarga que necesita el controlador tradicional del MVC a la hora de gestionar referencias a objetos y restaurar el estado de la aplicación.

En el controlador de la vista se escuchan los eventos y se ejecuta la lógica de forma similar al controlador del MVC. Una de las mayores diferencias entre ambos es que se debe crear un controlador de vista para cada parte independiente de la misma pantalla (igual que pasa con el modelo de la vista), mientras que en el controlador tradicional se controlan todos los elementos que constituyen la pantalla.

Representación del Modelo-Vista-ModeloDeLaVista de una aplicación y del controlador de la misma. Imagen extraída de:
http://202.202.5.145/extjs/docs/whats_new/5.0/whats_new.html



[Imagen 5]

El controlador de la vista y su modelo forman parte de los componentes del ciclo de vida de la vista, lo que significa que por cada instancia de una vista solo se debe configurar uno de cada. Del mismo modo, cuando una vista es destruida, como ambos están asociados a ella, ellos también son destruidos. Esto puede ser positivo o negativo según cómo se mire, ya que una aplicación (en teoría) libera memoria cuando destruye el controlador de la vista, debido a que es innecesario escuchar eventos de una vista que no existe; pero también es posible que crezca el tamaño de la memoria porque se pueden llegar a crear varios controladores y modelos de vistas por cada pantalla.

4.2. BUENA PRÁCTICA PARA ORGANIZAR LAS DISTINTAS VISTAS DE UNA PANTALLA

En este apartado se describirá una de las formas más utilizadas para estructurar todas las vistas de una aplicación web creadas con EXT JS. Para ello se comentarán los principales *widgets* que se pueden encontrar en la librería de componentes del framework y las diferentes formas en las que se pueden colocar.

4.2.1. CONTENEDORES

Dentro de la multitud de ítems que forman EXT JS 5, hay una serie de ellos que tienen como misión contener a otros componentes. Estos ítems tienen una propiedad llamada *"layout"* en la cual se define cómo se deben colocar los ítems que almacenan. Los dos principales contenedores son:

1. Ext.container.Container:

   Este contenedor de EXT se podría definir como una simple etiqueta *div* en html, ya que se emplea para definir un bloque de contenidos en el que se pueden aplicar diferentes estilos.

1. Ext.panel.Panel:

   El *panel* es un contenedor de ítems similar al *container*, debido a que su funcionalidad es la misma. La diferencia está en que este ítem no se traduce a un simple *div*, ya que posee una cabecera en la que se puede insertar un título y está capacitado para incluir en él botones del tipo *toolbar*.

Estas son las principales diferencias, pero siempre se pueden encontrar más consultando la API oficial.

4.2.2. ÍTEMS DE FORMULARIO

Una vez explicados los ítems contenedores, se pasará a detallar algunos de los componentes que más se usan en la arquitectura. Si se desea conocer las configuraciones, métodos, eventos y propiedades de estos ítems o de otros que no están descritos en este documento, con una simple consulta al API se obtendría toda la información.

Muchos componentes heredan las características de la clase básica *field* del EXT JS. Estos ítems se utilizan en las vistas de formularios o de detalle en las que el usuario introduce diferentes datos o espera que se rellenen con los datos provenientes del servidor. Como los valores pueden ser de diferentes tipos, hay distintos *field* según lo que más se amolde a cada campo:

- *numberfield*: dato numérico.
- *textfield*: para introducir un pequeño texto.
- *textareafield*: en este caso, el texto que introduce el usuario es mayor que en un *texfield*.
- *datefield*: al pulsar sobre este campo se despliega un calendario en el cual el usuario debe seleccionar la fecha.
- *timefield*: similar al *datefield*, pero en este caso se despliega un listado de opciones con las diferentes horas del día. Se puede modificar el formato horario en el que aparecen, la diferencia de tiempo que hay entre las opciones mostradas, etc.
- *radiobutton*: se proporciona una lista de opciones de la cual el usuario solo podrá seleccionar una.
- *checkbox*: es similar al *radiobutton*, pero en este caso el usuario puede seleccionar varias opciones.
- *combobox*: es otra forma de seleccionar un elemento de una lista. En este caso sólo aparece en la vista de la aplicación un campo vacío, en el cual al ser pulsado aparecen los distintos datos con los que rellenar el campo.

4.2.3. GRID

El componente *grid* es otro de los más destacados en EXT JS, ya que se trata de una tabla que al ser alimentada con un *store* se auto-cargan diversos datos.

Para que la tabla se rellene automáticamente, en el componente se debe definir la propiedad *bind,* para determinar el *store* del cual se tomarán los datos y que previamente estará definido en el *ViewModel* de la vista. Ya que el *store* está compuesto de varias instancias de un mismo modelo (el cual se especifica dentro del *store* característico del *ViewModel*), lo único necesario para el autorrellenado del *grid* es que el *name* del dato que se quiere que aparezca en una columna, sea idéntico al *dataIndex* que se define en la propia columna.

Otra de las cosas que se puede conseguir con un grid es que al pulsar una fila o una celda se produzca un cambio en la vista.

4.2.4. LAYOUT

Para configurar un layout debemos introducir el siguiente parámetro en nuestro componente contenedor:

```
layout: {
    type: 'vbox',
    align: 'stretch'
}
```

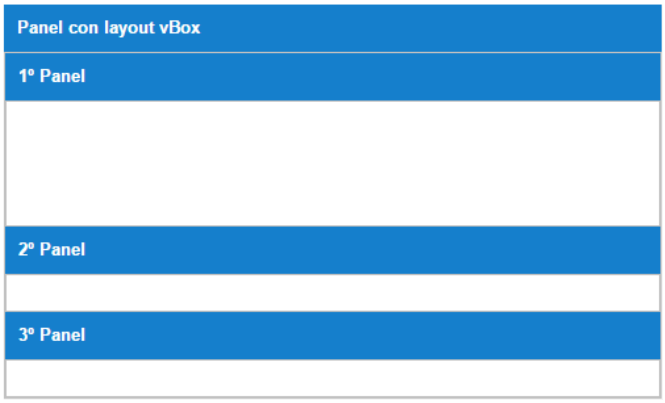
Con solo cambiar el atributo *type* del *layout* se consigue cambiar la colocación de los componentes de dentro del panel. El atributo *align: 'stretch'* indica que los ítems deben ocupar el 100% del tamaño reservado para ellos.

Si a la hora de insertar un ítem en el contenedor, se le añade el parámetro *flex*, se consigue que la proporción del tamaño de éste con respecto al total del contenedor, sea variable. Por ejemplo, se tiene un contenedor con una altura de 300 px que contiene 3 paneles. El primer panel tiene una altura fija de 200 px; los paneles 2 y 3 están configurados con *flex: 2* y *flex: 1* respectivamente. Esto hace que los dos últimos paneles se repartan el tamaño restante (100 px), de manera que la altura del 2º panel sea justamente el doble que la del 3º.

Las principales formas de organizar los diferentes ítems que se desean insertar en un componente contenedor son:

1. vBox: Coloca los ítems de forma vertical, uno debajo del otro, ocupando todo el ancho del contenedor. Hay que definir la altura reservada para cada componente para una correcta visualización.

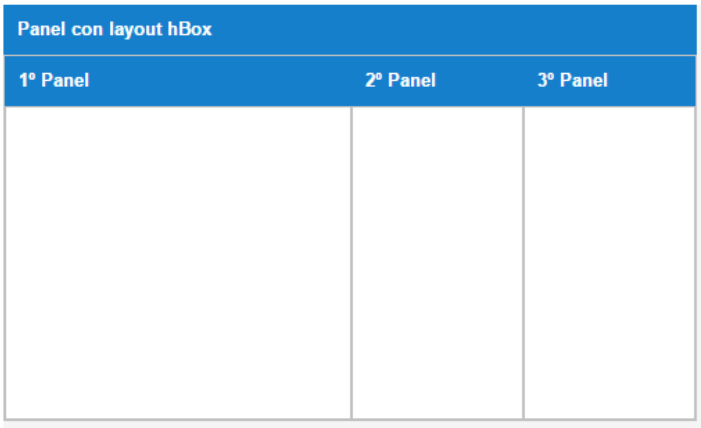
Panel principal con layout "vBox", el cual organiza los tres ítems que contiene.



[Imagen 6]

2. hBox: Este *layout* es similar al vBox, pero en este caso organiza los ítems de forma horizontal. Hay que definir el ancho de cada componente.

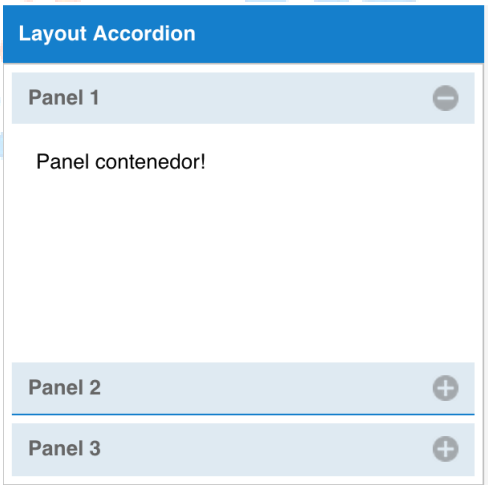
Panel principal con layout "hBox", el cual organiza los tres ítems que contiene.



[Imagen 7]

3. accordion: Esta forma de colocación interactiva de componentes es muy atractiva visualmente, ya que uno de ellos se muestra de forma completa, mientras que los demás se encuentran en forma comprimida.

Panel principal con layout "accordion", el cual organiza los tres ítems que contiene.

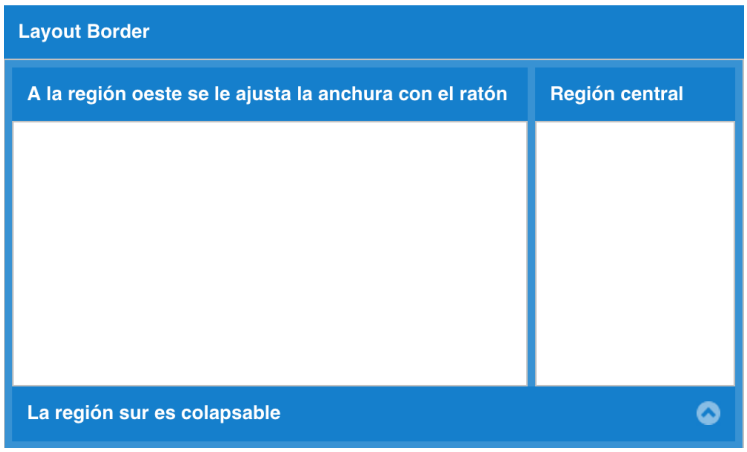


[Imagen 8]

4. border: Con este *layout* se pretende dividir el contenedor principal en: norte, sur, este, oeste y centro, siendo este último obligatorio. Para un correcto funcionamiento de *layout*, los contenedores del norte y sur deben tener una altura definida, mientras que para los del este y oeste se definirá la anchura. Una de las principales características de esta forma de organizar los ítems es que los contenedores de arriba, abajo, derecha e

izquierda podrán contraerse y cambiar su tamaño con un simple arrastre del ratón en el borde del contenedor.

Panel principal con layout "border", el cual organiza los tres ítems que contiene.



[Imagen 9]

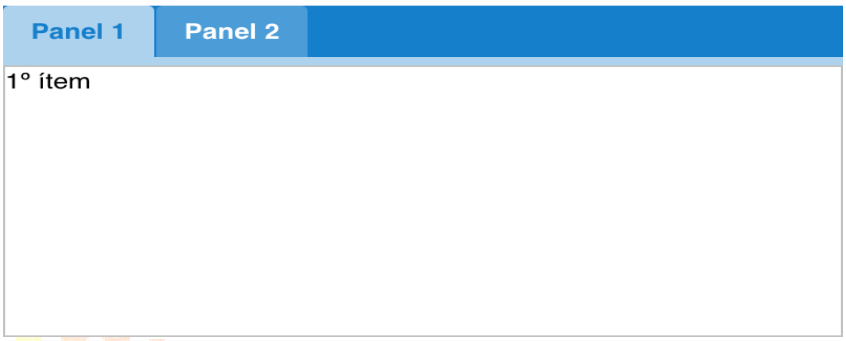
5. card: Este layout es especial, ya que únicamente se mostrará de forma completa el primer ítem que contiene. Para que se muestre otro componente, se debe interactuar con un botón, el cual se le programará para que al ser pulsado, se cambie el ítem a mostrar.

Panel principal con layout "card", el cual organiza los tres ítems que contiene.

[Imagen 10]

6. tabpanel: Esta forma de organizar los ítems no es un *layout*, pero funciona de forma similar al de tipo *card*; ya que al definir al contenedor principal, se creará una barra de herramientas automáticamente con tantas pestañas como ítems contenga el *tabpanel*, siendo el título de cada una de ellas el del propio ítem.

   Panel principal con layout "tabpanel", el cual organiza los dos ítems que contiene.



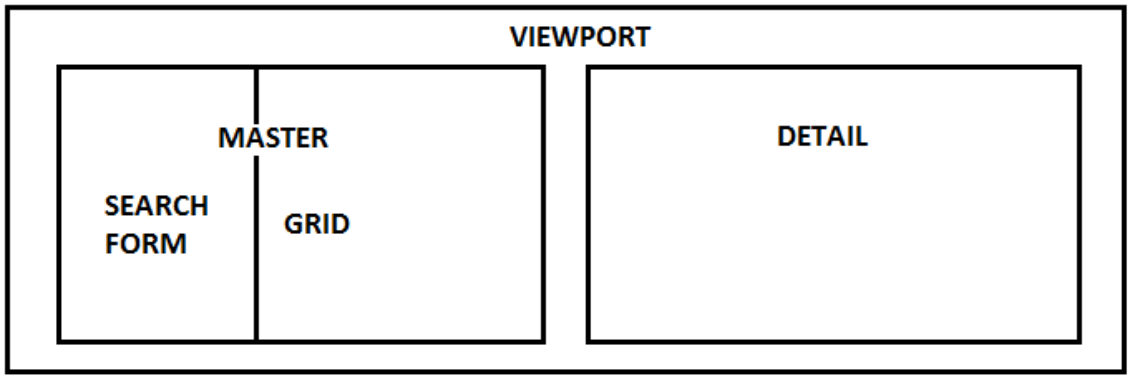
[Imagen 11]

4.2.5. ORGANIZACIÓN DE UNA PANTALLA

Una vez presentados los ítems contenedores y los principales *layout*, se describirá una manera en la que se pueden estructurar las diferentes vistas de una pantalla.

Las aplicaciones están compuestas por diferentes pantallas, que serán ítems contenedores de temáticas independientes. A su vez, la pantalla está formada por un *viewport*, que normalmente suele ser un *panel* de tipo *card*. Ésta será la vista principal de la pantalla y suele contener dos ítems, que nuevamente serán de tipo *panel*. Al primero se le conocerá como *master* o vista maestro y al segundo como *detail* o vista detalle. El *master* se suele diseñar como un *panel* con *layout* de tipo *border*, ya que en él se encontrarán nuevamente dos *paneles*: un formulario de búsqueda, conocido como *search form*, el cual se suele colocar en el oeste o en el norte; y un *grid,* en el que se mostrarán los resultados de la búsqueda. En el *detail* se deben mostrar los campos que detallan el modelo principal de la temática de la pantalla, los cuales estarán rellenos si el modelo ya existía en la base de datos o vacíos si se está ante una nueva creación. Se puede entender mejor con el siguiente dibujo:

Representación de una maquetación de una pantalla.



[Imagen 12]

Una vez vista la colocación de los elementos que componen nuestra pantalla, se detallará cómo organizar los diferentes ficheros que componen la aplicación:

Estructura de ficheros del Front-End de una aplicación creada con EXT JS 5.

[Imagen 13]

Se pueden observar varios detalles:

- En la carpeta app se encuentran clasificados los modelos y vistas (con sus *ViewModel* y *Controller*) que forman la aplicación.
- En la carpeta *data*, se encuentran los ficheros json de los cuales se cargan los datos de ejemplo. Sólo se deben cargar cuando la aplicación no esté conectada al Back-End, ya que cuando lo esté, será el servidor quien envíe los json reales.

Ejemplo de un archivo JSON.


```
{
    "data": [
        {
            "nombre": "KokeResurreción",
            "equipo": "AtléticodeMadrid",
            "pais": "España",
            "posicion": "MedioCentro"
        }
    ],
    "info": null,
    "success": true,
    "totalCount": 1
}
```


[Imagen 14]

- El archivo *index.html* será el primer archivo que se cargue al ejecutar la aplicación. El EXT JS usa el *Microloader* de este archivo para cargar la aplicación que se describe en el *app.json*, con lo que se consigue no tener que añadirlo directamente al *index.html*.

Ejemplo del index.html

```
<!DOCTYPE HTML>
<html>
<head>
    <meta charset="UTF-8">

    <title>MyApp</title>

    <!-- The line below must be kept intact for Sencha Cmd to build your application -->
    <script id="microloader" type="text/javascript" src="bootstrap.js"></script>

</head>
<body></body>
</html>
```

[Imagen 15]

- El fichero *app.json* es muy importante, ya que en él se incluyen todas las configuraciones de la aplicación, como por ejemplo: dónde se encuentra el archivo *app.js* y los archivos *css y sass*, cómo configurar los distintos modos de construcción de la aplicación (desarrollo, testing y producción), etc. Se puede encontrar en internet mucha más documentación acerca de lo que se puede incluir en él.

- El *app.js es una* extensión del archivo *Application.js*. En él se define cuál será la primera vista que se instanciará al iniciar la aplicación. El parámetro de configuración *mainView* es nuevo en la versión EXT JS 5.1 (hace lo mismo que *autoCreateViewport* de las versiones anteriores) y en él se hace referencia a la primera pantalla que se mostrará al iniciar la aplicación.

Ejemplo del app.js

```
Ext.application({
    name: 'MyApp',

    extend: 'MyApp.Application',

    mainView: 'MyApp.view.main.Main'

    //-------------------------------------------------------------------------
    // Most customizations should be made to MyApp.Application. If you need to
    // customize this file, doing so below this section reduces the likelihood
    // of merge conflicts when upgrading to new versions of Sencha Cmd.
    //-------------------------------------------------------------------------
});
```

[Imagen 16]

- Toda aplicación de EXT JS comienza con una instancia a la clase *Application*. Esta clase es lanzada al inicializar el *app.json*. En el *Application.js* se encuentran las configuraciones globales de la aplicación, como el *namespace*, *stores* globales, etc.

Ejemplo del Application.js

```
Ext.define('MyApp.Application', {
    extend: 'Ext.app.Application',

    name: 'MyApp',

    stores: [
        // TODO: add global/shared stores here
    ],

    launch: function () {
        // TODO - Launch the application
    }
});
```

[Imagen 17]

4.3. COMUNICACIÓN CON SERVIDOR

En este sub-apartado se explicará la tecnología que se utiliza principalmente en EXT JS para realizar las diversas peticiones al servidor.

4.3.1. TECNOLOGÍA AJAX

Para desarrollar aplicaciones con una mayor interactividad se utiliza la tecnología de comunicación con el servidor ***AJAX***.

En realidad, *AJAX* no es una tecnología en sí misma, ya que son varias tecnologías independientes que se unen y complementan a la perfección. Éstas son:

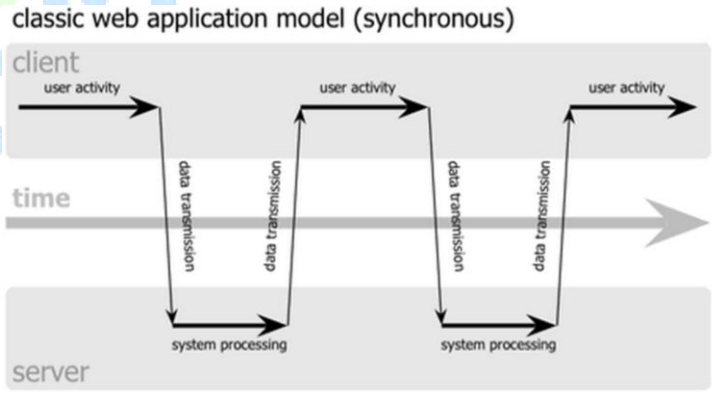
- XHTML y CSS: crea una presentación basada en estándares.

- DOM: interacciona y manipula dinámicamente la vista.
- XML, XSLT y JSON: para intercambiar y manipular información.
- XMLHttpRequest: intercambio asíncrono de la información.
- JavaScript: lenguaje en el que se implementa la lógica de presentación, dotando de interactividad a la aplicación y orquestando todos los elementos anteriores.

La principal ventaja de usar *AJAX* con respecto a las tecnologías tradicionales es que las acciones que un usuario realiza dentro de la aplicación web (pulsar un botón, realizar una búsqueda, seleccionar un valor de una tabla, etc.) generarán llamadas asíncronas, haciendo que el navegador continúe mostrando la página sin necesidad de recargarla. Los cambios se mostrarán una vez que se haya recibido la respuesta. Esto es posible gracias a que el intercambio de información con el servidor se realiza en segundo plano.

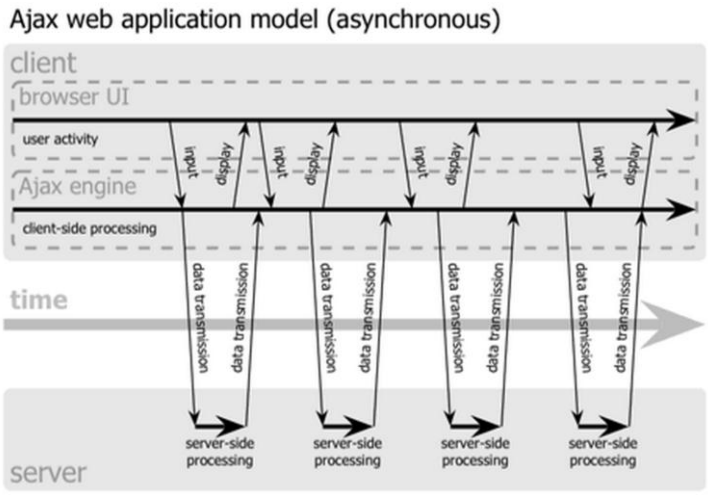
De este modo, la recarga constante de las páginas tras una comunicación con el servidor es eliminada, ya que *AJAX* crea un elemento intermedio entre el cliente y el servidor. Esta capa mejora la respuesta de la aplicación, debido a que el usuario nunca se encontrará con la ventana del navegador vacía, mientras espera una respuesta.

Diagramas de comunicación síncrona (aplicación web tradicional) .



[Imagen 18]

Diagramas de comunicación asíncrona (aplicación AJAX).



[Imagen 19]

Como se puede observar, las peticiones HTTP de la forma tradicional son sustituidas por peticiones JavaScript que se realizan al elemento *AJAX*. Las peticiones simples que no requieren la intervención del servidor, reciben una respuesta inmediata. Si se requiere de una respuesta del servidor, la petición se realiza de forma asíncrona mediante el *AJAX*. En este caso, la interacción del usuario no se verá interrumpida durante la espera de la respuesta.

Algunas de las aplicaciones web más conocidas basadas en *AJAX* son: Gmail, Google Maps y Google Docs.

4.3.2. AJAX EN EXT JS 5

En una aplicación creada con el framework EXT JS se suelen realizar peticiones *AJAX* cuando se desea recibir datos del servidor. Para ello, se debe enviar un formulario.

Para verlo más fácilmente, utilizaremos un ejemplo proporcionado por el API de EXT JS 5.

Ejemplo de petición Ajax realizado por en el Front-End.

```
Ext.Ajax.request({
    url: 'data/buscadordejugador.json',
    timeout:60*1000,
    params: {
        id: 1
    },
    success: function(response){
        // Proceso que se realizará si la petición ha sido exitosa
    }
    failure: function(response){
        // Proceso que se realizará si la petición ha sido fallida
    }
});
```

[Imagen 20]

Como se puede observar, esta petición debe ser realiza al pulsar el botón destinado a enviar los datos del formulario al servidor. Las propiedades básicas que se necesitan definir para realizar una petición son:

- url: dirección del servidor que procesa la petición.
- params: los distintos parámetros que se enviarán al servidor. Éste los recibirá como *RequestParam*.
- success y failure: en estas propiedades se define el método que se debe ejecutar si se ha recibido una respuesta exitosa o fallida del servidor.
- *callback*: en esta propiedad se define el método que se desea ejecutar una vez se haya procesado la respuesta del servidor.
- *timeout*: tiempo máximo (en milisegundos) en el que se espera una respuesta del servidor.

4.3.3. API REST

La API de REST permite crear servicios que pueden ser usados por cualquier dispositivo o cliente que entienda HTTP.

Para una correcta calidad a la hora de aplicar *REST* se debe tener en cuenta que:

1. Nunca se debe guardar el estado de la aplicación en el servidor, ya que toda la información que se requiere se debe proporcionar en la consulta que realiza el cliente.

   Gracias a esto, las aplicaciones son mucho más escalables, porque al no tener que almacenar variables de sesión, se podrán utilizar distintas tecnologías en un mismo recurso.

2. Mientras que en una aplicación tradicional cada URL permite acceder a una página distinta, en una en la que se utiliza *REST* cada URI (ya que son URLs identificativas) proporciona un **recurso** distinto en el servidor; por lo tanto este recurso es la información a la que se quiere acceder, modificar o borrar.

   Para que la URI sea correcta, no debe incluir verbos que indiquen acción. Ej: */futbolista/34*. La URI indica que se van a consultar, modificar o borrar los datos del futbolista con identificador número 34. También debe ser independiente del formato en el que se desean recibir los datos.

   Para filtrar resultados en *REST* se debe utilizar los parámetros de HTTP, como por ejemplo: */futbolista?equio=Real_Madrid&nacionalidad=Española.*

3. Para desarrollar una aplicación con *REST* se deben conocer:

   - **Los métodos HTTP**, ya que como en la URI no se debe especificar la acción que se realiza al acceder al recurso, será en la propiedad método donde se indicará.
     Los diferentes métodos son: *GET* (consultar y leer un determinado recurso. Si en la URI no se indica el identificador, se devolverá toda la lista de recursos), *POST* (crear recurso. En la URI no aparecerá el identificador del recurso, ya que aún no está creado), *PUT* (modificar recurso identificado en la URI), *DELETE* (borrar recurso determinado) y *PATCH* (modificar partes concretas de un recurso especificado en la URI).
   - **Los códigos de error y de estado** que se utilizan en *REST* son los mismos que en HTTP, ya que de este modo su identificación es más rápida para cualquier desarrollador.
   - Para definir el **tipo y el formato** en el cual se desean recibir los datos, se utiliza la propiedad *accept*. Como se puede indicar mediante prioridad el tipo de formato en el que se quieren recibir los datos, el servidor nos debe indicar, en el *Content-Type* de la respuesta, el formato en el que finalmente se recibirá.

4.3.4. COMUNICACIÓN REST EN EXT JS 5

Para realizar una correcta comunicación con el servidor utilizando EXT JS 5 y *REST* se debe configurar el modelo de una manera específica. En él, se debe definir el ***proxy*** de tipo *REST* y la url de la que se quiere obtener el recurso. A continuación, se encuentra un ejemplo de cómo configurar un modelo.

Ejemplo de definición de un modelo en EXT JS 5

```
Ext.define('User', {
    extend: 'Ext.data.Model',
    fields: ['id', 'name', 'email'],

    proxy: {
        type: 'rest',
        url : '/users'
    }
});
```

[Imagen 21]

Una vez entendida la configuración, se explicará cómo realizar los distintos métodos de *REST* con este framework.

Un objeto de tipo ***Ext.data.Model*** posee los métodos que realizan las distintas peticiones:

- **Crear**: Para realizar la petición *POST*, se utiliza el método *save()*.

Ejemplo de creación de un recurso en el servidor desde el cliente de la aplicación.

```
var user = Ext.create('User', {name: 'Ed Spencer', email: 'ed@sencha.com'});

user.save(); //POST /users
```

[Imagen 22]

- **Recuperar**: Como antes se explicó, al realizar una petición *GET* se obtienen, o todos los elementos de un recurso, o uno en particular; dependiendo de si se especifica el identificador o no. Para realizar esta consulta al servidor, existe el método *load()*. Hay que darse cuenta de que si se desea obtener todos los elementos del recurso, se debe cargar un *store*, mientras que si es un elemento determinado, se debe cargar un *model*.

Ejemplo de carga de un recurso (o varios) en el cliente desde el servidor.

```
//1. Load via Store

//the Store automatically picks up the Proxy from the User model
var store = Ext.create('Ext.data.Store', {
    model: 'User'
});

store.load(); //GET /users

//2. Load directly from the Model

//GET /users/123
User.load(123, {
    success: function(user) {
        console.log(user.getId()); //outputs 123
    }
});
```

[Imagen 23]

- **Modificar**: Para modificar un elemento, se debe cargar previamente. Una vez realizados los cambios, se llamará nuevamente al método *save()*, pero como ahora sí existe un identificador en el servidor para el elemento, se realizará una petición *PUT* en lugar de *GET*.

Ejemplo de modificación de un recurso en el servidor desde el cliente de la aplicación.

```
//the user Model we loaded in the last snippet:
user.set('name', 'Edward Spencer');

//In this case it will perform a PUT request to /users/123.
user.save({
    success: function() {
        console.log('The User was updated');
    }
});
```

[Imagen 24]

- **Borrar**: Al igual que para modificar, se debe cargar el modelo que se desea borrar. Una vez hecho, se llamará al método *erase()* para eliminar el elemento de la base de datos.

Ejemplo de eliminación de un recurso en el servidor desde el cliente de la aplicación.

```
//Performs a DELETE request to /users/123
user.erase({
    success: function() {
        console.log('The User was destroyed!');
    }
});
```

[Imagen 25]

4.4. GUÍA PARA LA CREACIÓN DE UNA APLICACIÓN CON EXT JS 5

Para crear el Front-End de una aplicación web con el framework EXT JS 5 se debe disponer de una serie de elementos:

- Descargar e instalar **Sencha CMD**. https://www.sencha.com/products/extjs/cmd-download/
- Poseer un **editor de texto**. (El autor de la memoria recomienda usar el editor *WebStorm*, ya que notifica los errores sintácticos de JavaScript que el programador pueda cometer).
- Descargarse el **framework EXT JS 5**. Lo podrá obtener de forma gratuita desde: https://github.com/thebohodon/ext.js.5.0.1-gpl
  Se debe saber que este framework es válido únicamente para realizar Front-End de aplicaciones web no comerciales. Si se desea comercializar se debe descargar la licencia oficial desde la página de Sencha.

Ahora se detallarán los pasos a seguir para la creación:

1. **Crear un workspace**: para ello se navega a través del terminal hacia la carpeta donde se desea ubicar el proyecto. Al escribir: ***sencha generate workspace <directorio_donde_ubicarlo>***, el *Sencha CMD* lo generará.

2. **Crear la aplicación**: una vez dentro del *workspace*, escribiremos en el terminal: ***sencha -sdk <ruta_carpeta_framework_ext> generate app <NOMBRE_APP> <directorio_donde_ubicarla>***. Esto generará todos los ficheros necesarios para que la aplicación pueda

correr sin problemas. Así mismo, se autogenera una vista principal con sus respectivos *ViewModel* y *ViewController*. Si se tiene el último *Sencha CMD* instalado, se podrá ejecutar en su lugar el comando: ***sencha generate app -ext <NOMBRE_APP> <directorio_donde_ubicarla>***. Esto descargará automáticamente el último framework disponible de EXT, que será con el que se generará la aplicación.

3. **Correr aplicación**: El programa *Sencha CMD* incluye un servidor web que escucha en el puerto: 1841de la máquina donde se ejecute. Para inicializarle, hay que ubicarse con el terminal en el directorio donde se ha definido que se encuentra la aplicación. Una vez allí, el comando: ***sencha app watch*** lo inicializará en la url http://localhost:1841.

   La característica más importante que este comando ofrece, es que siempre que se detecta un cambio en los ficheros de la aplicación, el servidor está a la escucha, lo que significa que solamente se necesitará recargar la página web para observar los cambios realizados en el código.

4.5. TEMAS VISUALES EN EXT JS 5

En este apartado se tratará de explicar cómo desarrollar un tema visual, que es el que dota del look&feel a la aplicación, dando los diferentes estilos a los diversos componentes de las pantallas. Para ello, lo más importante es tener instalado el "**Sencha Cmd**", ya que funciona como un compilador de **SASS**.

4.5.1. CONSTRUCCIÓN DE UN TEMA

Una vez construido el *workspace* de la aplicación y el programa en sí, se procederá a crear su tema. Para ello, habrá que generar su esqueleto con el comando: "sencha generate theme {nombre del tema}". Éste se ejecutará desde el directorio de la aplicación. Una vez hecho esto, aparecerá la estructura de nuestro tema en la carpeta *pakages/local/{nombre del tema}*.

El directorio lo constituye una serie de carpetas y ficheros, de los cuales, los más importantes son:

- "**package.json**" – Es el fichero en el que se encuentran todas las propiedades del tema. El *Sencha Cmd* crea el

paquete con ciertas dependencias que son requeridas como el nombre y la versión.

- "**sass/**" – Este directorio contiene todos los ficheros *sass* que componen el tema principal de la aplicación. Estos ficheros están divididos en cuatro secciones:
    - "**sass/var/**" – contiene las variables *sass* del tema.
    - "**sass/src/**" – se encuentran los ficheros con las reglas de estilo *sass* y los diferentes *uis*. Estos archivos pueden usar las variables definidas en "sass/var/".
    - "**sass/etc/**" – contiene funciones y estilos opcionales del tema.
    - "**sass/example**" – incluye los ficheros que son autogenerados con la creación del tema. En ellos, se encuentran los estilos por defecto que tienen los componentes de EXTJS, por lo que estos archivos nunca deberían ser borrados.

- "**resources/**" – aquí se deben incluir las imágenes y todo tipo de recursos estáticos que el propio tema requiera.

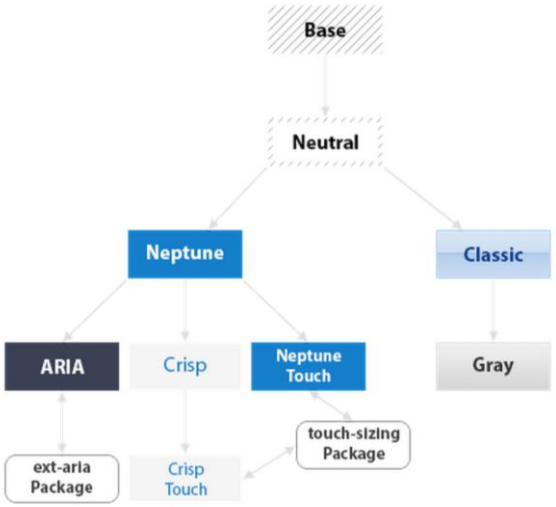
- "**overrides/**" – si se precisan archivos JavaScript, para que las clases de los componentes de EXT JS obtengan una correcta visualización, se deben incluir en este directorio.

Nota: Todos los ficheros y directorios en "sass/var/", "sass/src", y "overrides" deben estar estructurados de la misma forma en la que aparecen en la jerarquía dentro de la ruta "*ext/src*". Ej. Si se quiere crear un fichero en el que se definan las diferentes variables a las que todos los paneles de la aplicación puedan acceder, debe estar localizado en: "sass/var/panel/panel.scss".

4.5.2. CONFIGURACIÓN DE LA HERENCIA DE LOS TEMAS DE SENCHA

Inicialmente Sencha creó un tema, del cual se han ido desarrollando los sucesivos.

El árbol genealógico de los temas diseñados por los desarrolladores de la empresa Sencha. La imagen ha sido extraída de:
https://docs.sencha.com/extjs/5.1/core_concepts/theming.html



[Imagen 26]

Se deberá elegir el tema del cual heredará el que hemos creado. Para ello se deberá cambiar la propiedad *"extend"* del archivo *package.json* presente en el tema por el nombre del tema a heredar. Se recomienda que sea uno de estos: *"ext-theme-neptune"*, *"ext-theme-classic"* o *"ext-theme-crisp"*, ya que éstos contienen todo el código necesario para la creación de una interfaz atractiva.

4.5.3. CONFIGURACIÓN DE LA VISUALIZACIÓN DE LOS COMPONENTES

Todos los componentes en EXTJS poseen una serie de características visuales que pueden ser modificadas. Éstas están descritas en la documentación del framework, dentro de las opciones del menú *CSS Vars* y *CSS Mixins*.

Por un lado, en el primer menú se describen todas las variables que se pueden modificar. Por ejemplo: si se desea modificar el color de fondo con el que aparecerán todos los paneles de la aplicación, en el fichero "sass/var/panel/panel.scss" se deberá incluir: *$panel-body-background-color:* y el color que se desee.

En el segundo menú, se muestran los diferentes tipos de visualización que tiene por defecto un componente. Esto quiere decir, que a los componentes se les puede asociar durante su creación, un grupo de características visuales. Sencha pone algunos a disposición

del usuario, pero se pueden crear nuevos grupos. Un ejemplo de ello sería (situado en: "*sass/src/panel/Panel.scss*"):

```
@include extjs-panel-ui(
   $ui: 'highlight-framed',
   $ui-header-background-color: red,
   $ui-border-color: red,
   $ui-header-border-color: red,
   $ui-body-border-color: red,
   $ui-border-width: 5px,
   $ui-border-radius: 5px
);
```

4.5.4. ESTILOS PERSONALIZADOS PARA LAS VISTAS

Una vez explicada la manera de personalizar los componentes incluidos en la aplicación de forma genérica, se detallarán otras maneras de dar estilos, pero en este caso, serán específicos de un componente o de una vista concreta. Se podrían describir tres formas de hacerlo:

1. **Dar estilos a los componentes:** Todos los componentes de EXTJS cuentan con atributos de estilos para modificar su visualización. Esta forma es la menos aconsejada, y solamente recomendada cuando se quiere dar un estilo muy específico a un campo de la vista, ya que si queremos que varios se comporten de la misma forma, tendríamos que replicar el código para cada uno de ellos.
2. **Hojas de estilo CSS:** Como en todas las páginas web, se puede incluir en el *index.html* los ficheros *css* en los cuales daremos los diferentes estilos a la aplicación.
3. **Variación del tema para una pantalla:** Si el usuario se localiza en el directorio de la propia aplicación, se podrá dar cuenta de que existen carpetas con el mismo nombre que las que aparecen en el directorio del tema que se genera para la aplicación. El fin de cada una de ellas, es el mismo que las que se encuentran en el tema, pero con un matiz, en lugar de cambiar la forma de visualizar un componente para que se vea igual en toda la aplicación, el cambio sólo afectará a una determinada pantalla. Para que esto tenga efecto, la jerarquía que tienen que seguir tanto en "sass/var/", "sass/src", como en "overrides" debe ser la misma que la que aparece en la carpeta "*view*".

4.6. "HOLAFIFA": APLICACIÓN BÁSICA DE EXT JS 5

Una vez explicado cómo crear una aplicación web y cómo darla los diferentes estilos de visualización, el programador podrá empezar a desarrollarla.

En este apartado se describirán los pasos que se han seguido para realizar una aplicación básica. Si el lector desea descargar esta prueba, podrá hacerlo desde: https://github.com/thebohodon/HolaFifa-EXTJS-5. Ésta pequeña aplicación ha sido desarrollada por el autor de la memoria, con el fin de ayudar a los lectores a entender mejor la forma programar el Front-End de una aplicación con Ext JS 5.

4.6.1. EL SERVIDOR

Debido a que este trabajo de fin de grado trata sobre el framework EXT JS 5, este apartado estará dedicado a la explicación del front-end de la aplicación; aun así se comentarán algunos detalles que se deben tener en cuenta en la comunicación entre el cliente y el servidor.

Para comenzar se detallará a grandes rasgos la funcionalidad del servidor. Para HolaFifa no se ha creado una base de datos real, sino que se trabaja con ficheros *json*, los cuales serán leídos y modificados según las peticiones recibidas. Hay que tener en cuenta que debido a esto, el servidor no funcionará tan bien como funcionaría si se realizaran consultas verdaderas.

Los pros de hacerlo de esta manera son:

1. No será necesaria la instalación de una base de datos, la parte cliente ni el servidor.

2. Cuando se reciba una petición, se podrá comprobar directamente en el fichero *json* la modificación realizada, sin necesidad de realizar ningún tipo de consulta.

Mientras tanto, los contras son:

1. Las consultas complejas, como son los casos de peticiones a las que se asocian determinados filtros, no se producirán. En su lugar, se devolverán todos los datos sin filtrar. A pesar de saber esto, el front-end de la aplicación se ha desarrollado de forma en la que sí se envían los diferentes parámetros que el usuario introduce, para que los datos devueltos sean más específicos.

2. Como la base de datos la forman ficheros *json,* que se encuentran localmente en el ordenador personal del usuario, los cambios que en ellos se ocasionen, sólo serán efectivos en ese ordenador.

El servidor ha sido desarrollado también en el lenguaje JavaScript, siendo en este caso el framework utilizado NodeJs, con la versión de *Express4*.

Para las consultas, se dispone de unos servicios REST cuyas especificaciones son las siguientes:

- **GET /services/clubs**: Se encarga de devolver el listado completo de equipos.
- **GET /services/clubs/{Id}**: Devuelve el equipo con el id que case con el parámetro de url id.
- **GET /services/futbolistas**: Devuelve la lista de jugadores que han sido previamente añadidos.
- **POST /services/futbolistas**: Añade un nuevo futbolista, partiendo de los datos introducidos en la pantalla de detalle. El servidor se encarga de asignarle el id siguiente al último jugador creado.
- **GET /services/futbolistas/{Id}**: Devuelve el jugador con el id que case con el parámetro de url id.
- **DELETE /services/futbolistas/{Id}**: Elimina el futbolista con el id que case con el parámetro de url id de la colección.
- **PUT/services/futbolistas/{Id}**: Actualiza los datos del jugador con el id que case con el parámetro de url id.

Los datos que se manejan se extraen de los ficheros *json*:

- **/server/data/clubs.json**: Datos de los diferentes equipos a los cuales un jugador puede pertenecer.
- **/server/data/futbolistas.json**: Se trata de la colección de jugadores que se han añadido previamente.

Cada operación CRUD perdurará en su fichero JSON correspondiente.

Para arrancar el servidor, es necesario tener instalada la versión 0.10.0 de Node JS o superior en la máquina. Si no se tiene instalada, se deberá visitar la página http://nodejs.org/ para descárgala.

A continuación, es necesario instalar las dependencias del servidor (indicadas en el archivo /server/package.json). Para ello, es necesario situarse en esta ruta con la consola de comandos y ejecutar el comando "*npm install*". Esto, descargará las dependencias necesarias para correr la aplicación. Una vez hecho esto, dentro del directorio server se debe ejecutar el comando "*node server.js*" y ya estará arrancada la aplicación, escuchando por defecto en el puerto 8080. Si se quiere cambiar el puerto, se debe editar el archivo server.js y cambiar el valor de la variable port por el puerto que se quiera.

Toda la parte servidora ha sido realizada rápidamente gracias al magnífico trabajo de Manuel de la Vega, ya que para desarrollar este servidor se ha consultado el proyecto que él subió a su gitHub: https://github.com/madelavega/KataExtJS5.

4.6.2. EL CLIENTE

Las funcionalidades del cliente son las siguientes:

- Una pantalla, en la cual se muestra un buscador de jugadores, y una tabla con distintas columnas, en las que se representan características de un jugador. La tabla, se rellenará con los datos presentes en el fichero *json* que el servidor se encarga de proporcionar.

- Otra pantalla, a la que se accede al pulsar doble clic sobre un jugador representado en la tabla descrita anteriormente, o seleccionando un futbolista y tocando el botón editar. En esta vista, se mostrarán los distintos campos que detallan al jugador, siendo algunos editables. Si se pulsa el botón "Nuevo", también se accederá a la misma pantalla, pero en este caso, todos los campos aparecerán vacíos. Algunos

deben de estar rellenos para poder dar de alta a un nuevo futbolista.

En la carpeta *font-awesome-4.4.0* se encuentran los *css* para dar los estilos de los iconos presentes en la aplicación. Si se visita la web https://fortawesome.github.io/Font-Awesome/ se podrán observar todos los iconos que están disponibles.

Como la aplicación no ha sido desarrollada con "*sencha cmd*", ya que se deseaba que la aplicación la ejecutara el servidor *Node* y no el "*sencha watch*", no se han podido dar diferentes estilos mediante sass. En su lugar, se ha incluido el fichero *FifaStyle.css* en el cual se modifica la forma de visualizar los distintos elementos.

4.6.3. EL FRONT-END DE LA APLICACIÓN

HolaFifa sigue la misma estructura que la mostrada en la imagen 13.

En el fichero *index.html* se han añadido los links necesarios para que la aplicación pueda: utilizar los iconos *FontAwesome*, añadir un estilo de texto proporcionado por Google, usar la hoja de estilos que modifica la visualización de las vistas e insertar una imagen en el título de la pestaña de la web en el explorador.

Como se desea que todos los detalles que el framework EXTJS5 muestra por defecto estén en español (nombres de los meses en los calendarios, etiquetas que informan de campos obligatorios…), se ha requerido el paquete "*ext-locale*" y se ha especificado el idioma con "*locale*": "*es*" en el fichero *app.json*.

En la aplicación, están presentes dos modelos: el ***modelo club*** representa a un equipo al que un jugador puede pertenecer. Se caracteriza por tener asociado un identificador, un nombre y el nombre del país al que pertenece. El **modelo futbolista** cuenta con un identificador, el nombre del jugador, la nacionalidad, el identificador del equipo al que pertenece y el nombre del mismo, los años que lleva en el club, la fecha del cumpleaños y un flag para saber si ha ganado el balón de oro o no. En los dos últimos atributos, se les ha incluido un pequeño fragmento de código para obtener unos resultados concretos. Como el tipo del atributo cumpleaños es "*date*", se le ha especificado el formato de la fecha con la que se debe trabajar. Éste es: "*día-mes-año*". El día y el mes tendrán siempre 2 cifras, mientras que el año

constará de 4. Para el campo *balonoro*, el tipo que se le ha designado es "*boolean*". Ya que el dato que viene del servidor para este atributo puede ser *true* o *false,* se ha programado un método que transforma el valor al "*string*" "Sí" o "No" según lo que se reciba. En ambos modelos, el proxy utilizado para las comunicaciones con el servidor es de tipo *rest*.

Las vistas de la aplicación se estructuran de la siguiente manera:

***ViewPortMain*:**

Es la vista principal del programa, ya que él se encarga de albergar todas las demás pantallas. El *layout* que maneja los ítems que tiene a su cargo (vista maestra y vista detalle) es de tipo *card*, con lo que se consigue que inicialmente sólo se muestre el primero de ellos, y vayan alternándose dependiendo de las acciones del usuario. También contiene una barra de herramientas que presenta varios botones, los cuales, al ser pulsados, ejecutarán una función. Dependiendo del ítem que muestre el ViewPortMain, se visualizarán unos botones y se ocultarán otros. También puede que aparezcan desactivados si para realizar la acción que tienen asignados, necesitan que un elemento de la tabla esté seleccionado.

Los botones son:

- **Nuevo**: Muestra la vista del detalle. Los campos que aparecen están vacíos y se asocia un nuevo modelo, de la entidad futbolista, al formulario que forma la pantalla. Se muestra tanto para la pantalla maestra como en la detalle, y siempre se encuentra habilitado.

- **Editar**: Activa el segundo ítem del ViewPortMain. Se diferencia del botón nuevo, en que en este caso los campos del formulario aparecen rellenos con los datos provenientes del servidor. Sólo aparece si se muestra la pantalla maestra, y para que se habilite, se debe seleccionar un futbolista de la tabla.

- **Borrar**: Al pulsarlo, aparece una ventana en la que se pide una confirmación, ya que si se acepta, se realizará una petición al servidor para que elimine los datos del futbolista. Aparece siempre, pero dependiendo del ítem del ViewPortMain activo, varía un poco su funcionamiento; ya que si es la maestra la que está visualizándose, solo estará habilitado si hay un futbolista seleccionado; mientras que si la mostrada es la vista detalle, siempre aparecerá habilitado, y si la petición *delete* se realiza, se volverá a la

vista maestra.

- **Volver**: Como su propio nombre indica, ejecuta la función de volver a mostrar la vista maestra, por lo tanto sólo debe aparecer si se está mostrando el detalle de un futbolista.

Todos los métodos que llaman los botones están programados en el ViewPortMainController.

Una de las operaciones que requiere una especial mención por combinar un método de EXT y uno de JavaScript es:

```
Ext.Array.each(btns, function(bt){
  this.bottonView[layout.getActiveItem().xtype].apply(this,[bt]);
} ,me);
```

El *Ext.Array.each()* es una función creada por Ext que recibe dos parámetros: el primero, es un *array;* y el segundo, es una función que se ejecutará por cada posición del *array*; el valor de esa posición, es el parámetro que recibe la propia función. El tercer parámetro, será el ámbito que se le da al método. Esto significa que dentro de una función, el valor de la variable *this* no será la propia función, sino una referencia del objeto que se pasa en este último parámetro. En este caso, será una referencia al ViewPortMainController; por lo tanto, dentro del método de Ext se podrá acceder a cualquier atributo del controlador.

De forma similar funciona la función *apply()* de JavaScript. En este caso, el método sirve para indicar a la función que le llama, qué objeto será su ámbito y qué parámetros recibirá la función. Los parámetros deben formar parte de un *array*.

En JavaScript hay otra función que funciona del mismo modo que *apply(),* se trata de la función *call()*. La única diferencia, es que en el último método, cada parámetro que se quiera pasar a la función que le llama, serán los sucesivos parámetros que tiene el método *call()*, sin contar el primero, ya que igualmente éste se trata del *scope*. Ejemplo:

```
var Pedro= {
  nombre: "Pedro",
  presentacion: function(n1,n2){
        console.log('Me llamo' + this.nombre + n1 + n2);
  }
};

var fn=Pedro.presentacion;
```

presentacion.call(Pedro, 'Castro', 'Romero');
presentacion.apply(Pedro, ['Castro','Romero']);

Ambos métodos mostrarían el mismo mensaje en la consola de comando: "Me llamo Pedro Castro Romero".

A pesar de que el ViewPortMain tiene asociado un ViewModel, en este caso, no se le utiliza.

Master:

La vista maestra está compuesta de una tabla y un buscador. Para organizar el código de una forma más clara, se ha decidido definir cada ítem del *master,* en un fichero independiente.

Para esta vista, se ha decidido escoger un *layout* de tipo *border*. En la derecha se encuentra el buscador, mientras que en el centro se sitúa la tabla con los resultados de la búsqueda.

El buscador tiene una anchura determinada, la cual puede ser modificada por el usuario desplazando la barra que separa ambos ítems o pulsando el botón de minimizar. Esto se consigue con los atributos "*split: true*" y "*collapsible: true*" respectivamente. Como se puede observar, la tipografía de letra que tienen las etiquetas de los filtros, es la proporcionada por Google. Este estilo de letra se ha conseguido al asignar a todos los textos el valor "*font-family: 'Indie Flower',cursive*" desde el archivo "*FifaStyle.css*". Los botones "*limpiar*" y "*buscar*" hacen lo que su nombre indica. Los iconos se consiguen gracias al atributo "*glyph*", que presentan los paneles y botones. El combo que indica el equipo por el que se quiere filtrar a los jugadores, se rellena con los datos procedentes del archivo "*clubs.json*". El formato en el que tiene que aparecer la fecha en el campo "*cumpleaños*" vuelve a ser "*día-mes-año*".

La tabla ocupará el espacio restante de lo que ocupe el formulario en cada instante. Cuenta con 6 columnas, las cuales representan valores del detalle de un jugador. Algunas tienen un tamaño fijo, mientras que otras varían según el espacio restante. Tanto en la columna "*años*" como en la de "*¿balón de oro?*", se ha programado una función que se ejecutará antes de que se rellenen. Esto se realiza para que no se muestre directamente el valor que proviene del atributo del fichero *json*. En la primera columna mencionada, se calcula los años que tiene el jugador, en lugar de mostrar la fecha de nacimiento. En la segunda, se muestra el icono del balón de oro si el jugador lo ha conseguido.

Para que la aplicación sea consciente de cuándo un jugador de la tabla ha sido seleccionado o se le ha hecho doble clic, se tiene que añadir "*listeners*" a la propia tabla.

Las funciones que ejecutan los botones y los escuchadores están definidas en el MasterCtrl.

En el MasterVM se definen los *stores* que se cargarán en la vista: el de *clubs* para rellenar los valores del combo que selecciona equipos, y el de *futbolistas* para obtener la lista de jugadores que están presentes en el fichero "*futbolistas.json*". Para relacionar el *store* con el componente al que debe aportar los valores, se utilizan los *bind*. En los *stores,* también se pueden programar "*listeners*". En este caso se ha programado para que cuando cambie el valor del *store* futbolistas, se inserte en el título de la tabla, el número de datos que están presentes en ella.

Detail:

La vista detalle está compuesta por un formulario con varios campos que estarán vacíos, si se accede a la vista tras pulsar el botón "nuevo"; o rellenos, si se ha seleccionado un jugador de la vista maestra y se ha hecho doble clic sobre él o se ha pulsado el botón "editar".

Para crear un nuevo jugador será obligatorio introducir su nombre, nacionalidad y fecha de nacimiento, mientras que si se está editando, estos campos no podrán ser modificados.

El botón "guardar" permanecerá inactivo hasta que los campos obligatorios tengan datos. El botón "resetear" dejará los campos editados tal cual se encontraban cuando el usuario entró en la pantalla del detalle, siempre y cuando no se haya guardado, ya que lo que realmente hace este pulsador, es recargar el modelo que se muestra por pantalla.

En el DetailCtrl, se encuentran los métodos que realizan los botones de esta vista y las funciones para cargar, y persistir el detalle de los futbolistas.

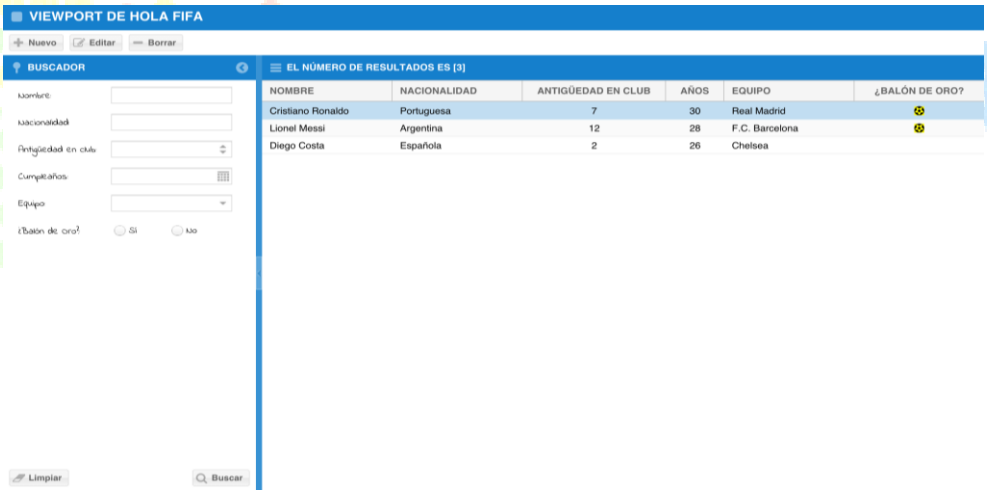
En el DetailVM, se define el *store* de clubs. Se realiza de la misma forma que en el MasterVM. En esta pantalla, no se tiene en cuenta el otro *store*, ya que en este caso solo necesitaremos cargar un modelo futbolista y no varios a la vez como sí sucede en la vista maestra.

Algunas imágenes que muestran el Front-End de la aplicación HolaFifa son:

ViewPort y Master inicialmente.

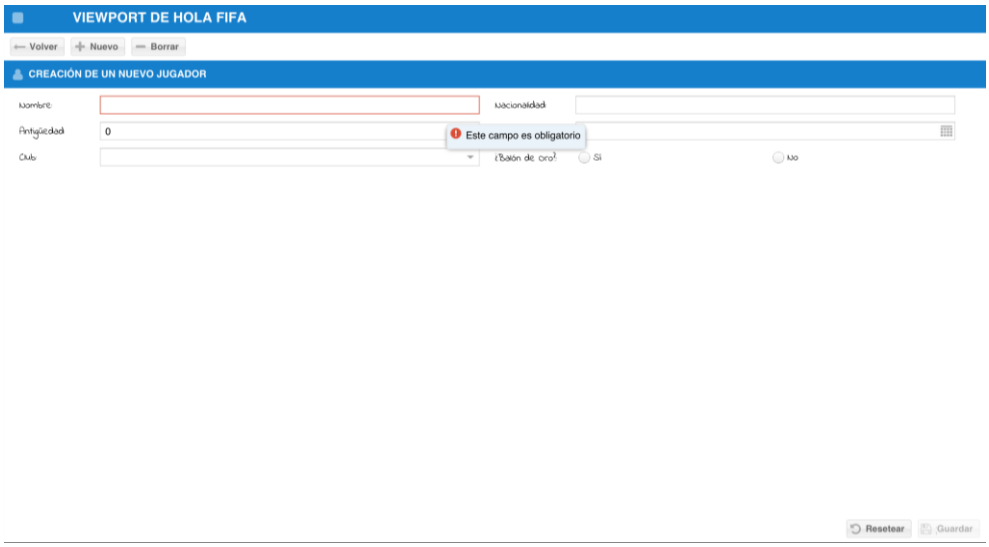
[Imagen 27]

ViewPort y Master. Se ha seleccionado un jugador de la tabla.



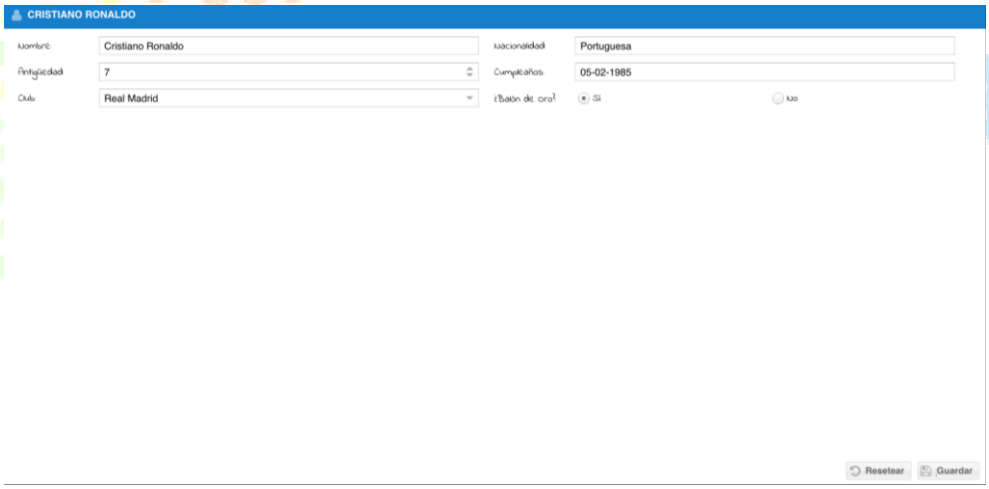
[Imagen 28]

ViewPort y Detail en la creación de un futbolista. La aplicación resalta los campos obligatorios.



[Imagen 29]

ViewPort y Detail en la modificación de un futbolista. Como el valor de la fecha del cumpleaños no se puede modificar, no aparece el icono del calendario.



[Imagen 30]

5. INTERFÁZ GRÁFICA COMPLEJA: MONITORIZACIÓN DE TRANSMISIONES DE DOCUMENTOS BANCARIOS

En este capítulo, se detallará en lo que consiste realmente el proyecto que se ha llevado a cabo. Se explicarán las distintas pantallas que el cliente ha demandado, cómo han sido maquetadas y también se expondrán imágenes en las que se visualicen los resultados.

Debido a que el proyecto ha sido realizado en la empresa **Indra Sistemas**, ésta se ha reservado varios derechos de imagen, por lo que no aparecerán fragmentos de código ni datos referentes al cliente destinatario de la aplicación.

Al tratarse de una aplicación real, el tiempo de finalización estimado en un principio no se ha correspondido con el que verdaderamente se ha dado, ya que inicialmente iba a consistir en una serie de pantallas simples, y durante el desarrollo del mismo se han producido diversos cambios de alcance aumentando tanto la complejidad de la aplicación como el número de funcionalidades a desarrollar.

En una empresa en la que trabajan miles de personas hay mucha gente involucrada en los proyectos, desde comerciales y gerentes, a planificadores y programadores.

Para este caso, las personas encargadas de conversar con el cliente, tienen la finalidad de averiguar qué es lo que éste realmente desea; después deben ponerse en contacto con el personal de planificación de proyectos, los cuales han generado un funcional en el que se especifica a los programadores cómo debe comportarse la aplicación web. Los informáticos serán finalmente los diseñadores del propio programa, ya que los de Front-End se encargan de maquetar las diferentes pantallas que el cliente ha pedido, y los del Back-End de gestionar y realizar las consultas a la base de datos.

Ambos equipos deben estar en continua comunicación, ya que si no la aplicación web no funcionaría correctamente; el servidor no recibiría los parámetros con los que el usuario desea realizar una consulta a la base de datos, ni tampoco se mostrarían por pantalla los datos que éstas generen. Todas estas reglas que se deben seguir entre ambas partes se conocen como *parámetros de negociación*.

La comunicación cliente-servidor siempre sigue las reglas de REST. Cuando hay envíos de contraseñas, por la red circulan cifradas, por lo

que ambos equipos saben cómo codificarlas y descodificarlas al enviarlas y recibirlas.

5.1. LOGIN Y MENÚ

Especificaciones del cliente

Para comenzar, como es lo más normal en cualquier aplicación web, se muestra una pantalla de identificación de usuario. En ella se debe introducir el nombre de usuario y la contraseña. Si dicho usuario se encuentra registrado en la base de datos, se cargará la vista principal de la aplicación.

Una de las características de la aplicación es que dependiendo del rol del usuario que se ha logeado, se mostrarán unas opciones de menú u otras.

Desarrollo Front-End

Esta vista se ha creado de forma independiente al resto de la aplicación. Se trata de una página en formato *jsp* que presenta un formulario en el que se pide introducir el nombre, y la contraseña del usuario.

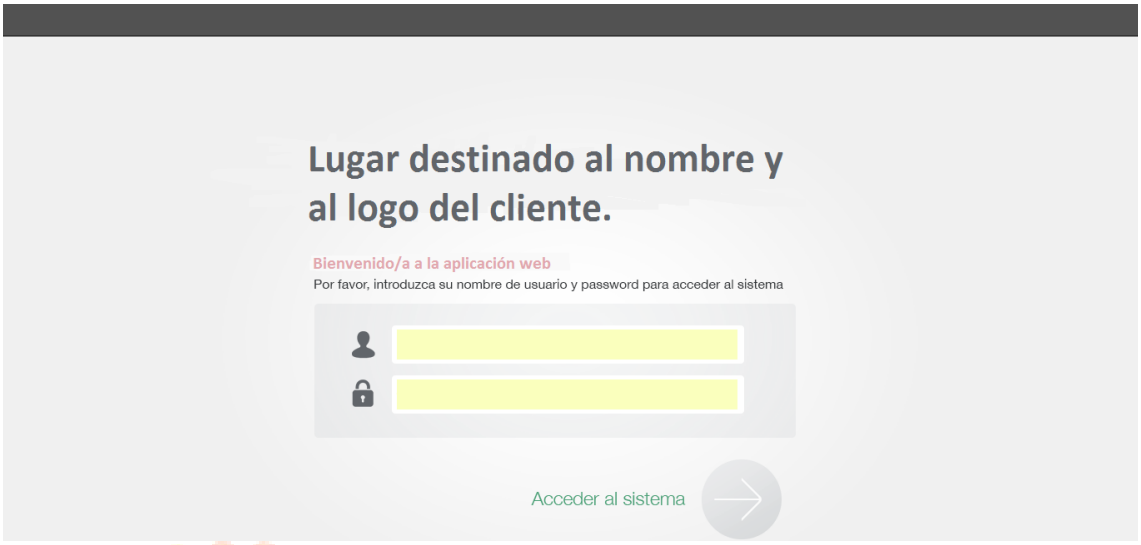
Cuando los datos llegan al servidor, éste se encarga de verificar que el usuario está dado de alta en la base de datos. Si lo está, el back manda una redirección al navegador web. La página destino será la url del *index* de la aplicación creada con EXTJS 5. A continuación, el back envía al front un archivo *json* con las opciones del menú a las cuales dicho usuario puede acceder. Éstas varían según el rol que tiene asignado el usuario en la aplicación.

El menú está formado por opciones, las cuales a su vez despliegan un submenú. Cada sub-opción se corresponde a una vista diferente de la aplicación. El menú está compuesto por:

- Auditoria: consulta de documentos y de transmisiones.
- Administración: usuarios, formatos de documentos, políticas de retención, tipos de documentos, parámetros globales, calendarios con los días inhábiles, instituciones o emisores de los documentos, sectores, contrataciones, instalaciones, workflows y reglas de procesamiento de documentos.
- Cuadro de mandos.

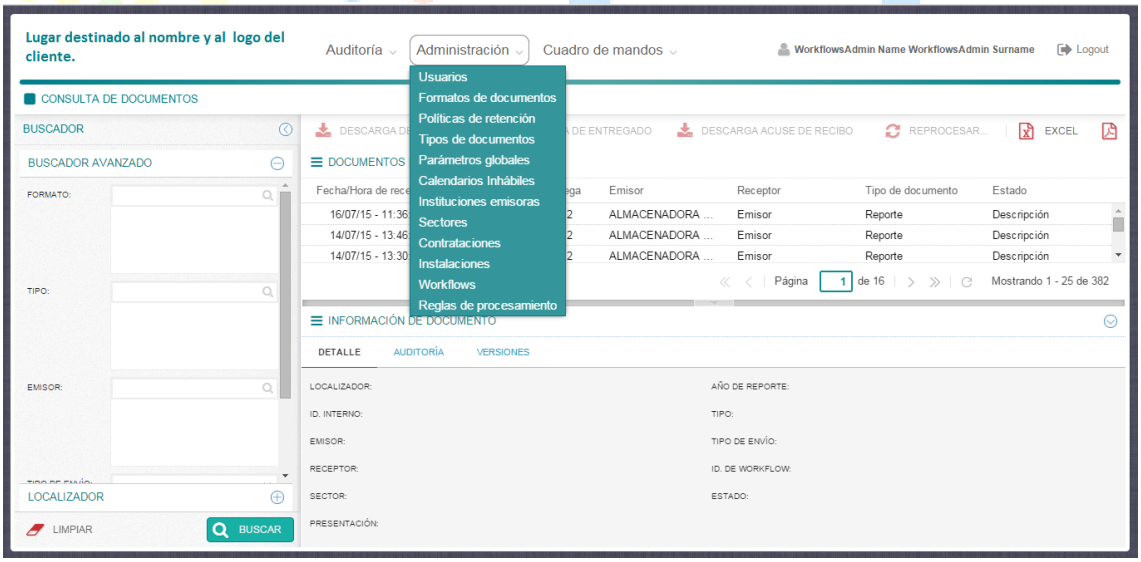
Visualización

Pantalla de Login



[Imagen 31]

Menú. Todas las pantallas de la aplicación lo muestran igual, pero para las siguientes imágenes sólo se adjuntará lo característico de la propia pantalla.



[Imagen 32]

5.2. LOGOUT

Especificaciones del cliente

Se mostrará una pantalla de final de sesión siempre que el usuario salga de ella de forma segura, o se agote el tiempo de inactividad con la aplicación.

Desarrollo Front-End

En este caso, la página HTML que muestra el navegador es la que aparece por defecto ante un código de respuesta HTTP 404 proveniente del servidor. Se llega a ella mediante cualquiera de los siguientes procedimientos:

- El usuario pulsa el botón "Logout", el cual está visible en cualquier punto de la aplicación.
- Automáticamente, si finaliza el tiempo de sesión.
- El usuario elige no seguir trabajando en la ventana que se muestra segundos antes del cierre de la sesión.

Visualización

Página de terminación de sesión.

[Imagen 33]

5.3. CIERRE AUTOMÁTICO DE SESIÓN

Especificaciones del cliente

Se debe avisar al usuario de que el tiempo de su sesión caducará próximamente.

Desarrollo Front-End

Para programar este funcionamiento de la aplicación se ha llegado a un acuerdo previamente con el equipo de Back, ya que son ellos los encargados de manejar el tiempo de sesión. Cada vez que el Font recibe una respuesta, lee de ella una cabecera especial que ha sido insertada por el servidor. En ella, se especifica el tiempo que le queda a la sesión.

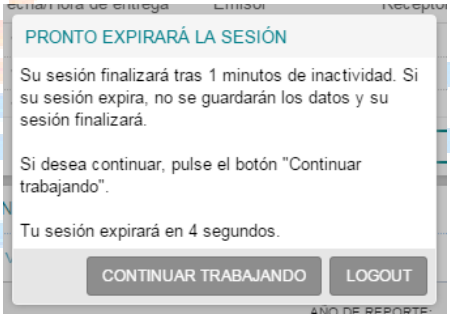
Se crea un temporizador con el valor recibido en la respuesta menos un minuto y veinte segundos. Éste es reseteado cada vez que se lee la cabecera. Cuando el contador salta, se comprueba si el usuario ha interactuado con la aplicación desde la anterior vez que se reseteó el temporizador (si ha movido el ratón o si ha pulsado el teclado). Si se ha detectado la presencia del usuario, se envía una petición

"fantasma" cuya finalidad es renovar la sesión. Si no ha habido interacción, se muestra una ventana de aviso con una cuenta atrás de un minuto. En dicha ventana aparecen dos botones, uno para seguir trabajando (envía la misma petición "fantasma" anteriormente comentada) y otro para finalizar la sesión. Si el temporizador llega a cero, se finaliza automáticamente.

Pero ¿por qué restar un minuto y veinte segundos al tiempo que se recibe en la cabecera de la respuesta, y no el minuto exacto que correspondería al momento en el que realmente finaliza la sesión? Por eso mismo no se hace así, ya que si se realiza la petición para finalizar la sesión cuando ésta ya ha acabado, el navegador recibirá un mensaje de código de error 500 (error interno ajeno al servidor), en lugar de redirigir al HTML que se ha desarrollado para este caso. De esta forma, se ha asegurado que todas las peticiones que la aplicación genera llegan al servidor.

Visualización

Avisador de final de sesión.



[Imagen 34]

5.4. MODIFICACIÓN DE DATOS DEL USUARIO

Especificaciones del cliente

Desde cualquier vista, el usuario registrado puede ser capaz de modificar sus datos. Éstos son: nombre, apellidos, e-mail, y contraseña.

Desarrollo Front-End

Cuando el usuario desea modificar alguno de sus datos personales, simplemente debe pulsar en la parte de la pantalla donde aparece su nombre. En ese instante, aparecerá una ventana manejada por un *layout card*. En el primer ítem aparecen los datos del nombre, apellido e e-mail que tiene actualmente el usuario, mientras que en el segundo se muestran tres campos vacíos: contraseña antigua, nueva y

confirmación. El botón "guardar" es diferente según el ítem activo de la ventana, por lo que se pueden guardar los datos de éste sin ser necesario el almacenamiento de los campos del ítem no visible.

Visualización

Modificación de datos de usuario.

Nombre, apellido y email.

[Imagen 35]

Contraseña.

[Imagen 36]

5.5. CONSULTA DE DOCUMENTOS

Especificaciones del cliente

Se desea una pantalla que muestre los diferentes documentos que han circulado a través de la red bancaria. Para que los resultados que generan la búsqueda sean menores, se podrá filtrar por localizador o por un grupo de características (formatos, tipos, emisores, tipo de envío, fecha inicial y final de recepción (se puede acotar también por horas) y estado).

Desarrollo Front-End

Esta vista es bastante compleja, ya que se compone de un buscador de documentos, que está colocado a la derecha, y de varias tablas que muestran la información de los resultados de la búsqueda, situadas en la parte central de la pantalla.

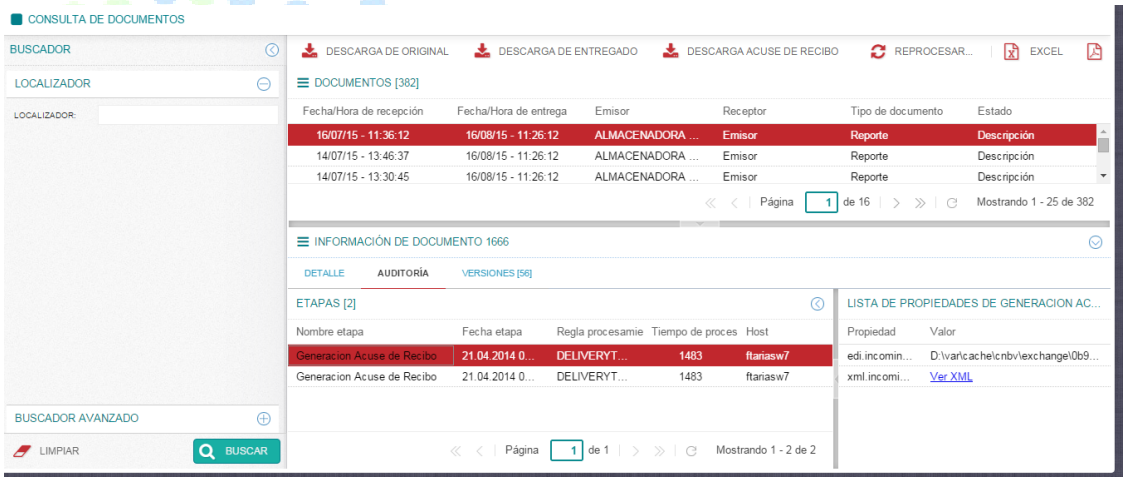
Para que el buscador envíe al servidor el localizador del documento o el grupo de características, se ha utilizado un *layout accordion*. Los filtros que se envían son únicamente los que pertenecen al ítem no colapsado. Para saber de qué contenedor obtener los parámetros de búsqueda, antes de realizarla se observan todos los ítems uno por uno, hasta localizar el que tiene la propiedad *collapsible* a *false*, de ese es del que se obtienen los filtros.

El detalle de los documentos mostrados en la pantalla es complejo, ya que estos presentan varias relaciones "1 a n". La parte central de la vista está dividida en dos, en el centro se encuentra una tabla con todos los resultados de la búsqueda y en el sur, el detalle específico del documento seleccionado en el *grid*. Debido a que la información que posee cada documento es muy extensa, se ha decidido utilizar un *layout tabpanel*. En el primer panel se muestran lo datos planos del documento. El segundo está formado por dos tablas, las cuales se rellenan en cascada. Al seleccionar un documento, se cargarán todas las etapas por las que ha ido pasando. Si se pulsa sobre una de ellas, se visualizarán en la tabla contigua todas las propiedades que esa etapa tiene asociadas. Finalmente, el tercer panel lo compone otro grid que muestra las diferentes versiones que se han generado del documento.

Al seleccionar un documento, se habilitan las opciones de descargar el original, el entregado y/o el acuse de recibo; también se puede reprocesar el documento. Los botones de descargar la lista de los elementos que aparecen en la tabla de resultados de la búsqueda, en Excel o en PDF, siempre aparecen habilitados, ya que antes de visualizar la pantalla, se ha realizado una búsqueda de todos los documentos que se han recibido dos días antes del actual.

Visualización

Buscador con el filtrado por localizador y detalle de las etapas del documento y de las propiedades de la etapa seleccionada.



[Imagen 36]

5.6. CONSULTA DE TRANSMISIONES

Especificaciones del cliente

Se deben monitorizar todas transmisiones de documentos. Se podrán filtrar los resultados de la búsqueda por: fecha inicial y final, categoría, canal e identificación de sesión.

Desarrollo Front-End

Esta pantalla es la más simple de la aplicación, ya que únicamente cuenta con un buscador y un listado de resultados.

Visualización

Pantalla de transmisiones.

[Imagen 38]

5.7. MANTENIMIENTO DE USUARIOS

Especificaciones del cliente

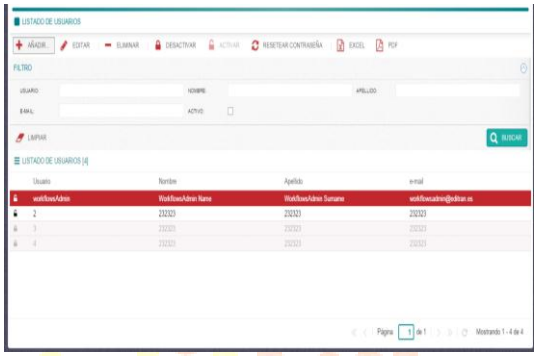
Para poder dar de alta a un nuevo usuario se debe hacer desde esta pantalla. Se podrá visualizar una lista con todos los usuarios que hay en la aplicación y a cada uno se le asignará un determinado rol. Los usuarios una vez dados de alta, podrán estar activos o no. También se proporcionará la opción de resetear la contraseña del usuario seleccionado.

Desarrollo Front-End

En este caso, la estructura de la pantalla es la de un viewport que contiene una vista maestra, con su buscador y su lista, y una vista detalle. Los usuarios no activos, se muestran sombreados. Se podrá cambiar esta propiedad desde la vista maestra sin tener que navegar al detalle del usuario.

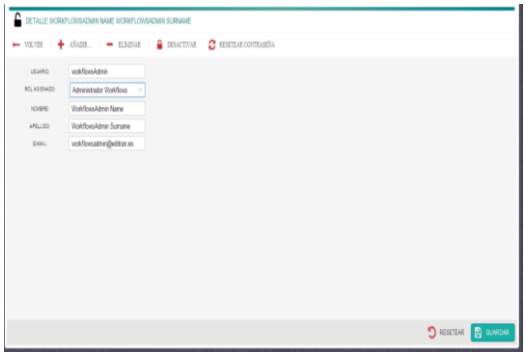
Visualización

Vista maestra.



[Imagen 39]

Vista detalle.



[Imagen 40]

5.8. ADMINISTRACIÓN DE MANTENIMIENTOS SIMPLES

Especificaciones del cliente

Para administrar los distintos mantenimientos que proporciona la aplicación, el cliente ha pedido varias pantallas. Todas son muy similares, ya que se pide un buscador para el mantenimiento y una lista de resultados. Se podrán dar nuevas altas, modificaciones y eliminaciones, pudiendo siempre descargar los resultados en formatos PDF y Excel.

Los diferentes mantenimientos con sus parámetros de búsqueda son:

- **Formato de documento**: Descripción y nombre del formato. En la vista de detalle, se debe dar la opción de subir al servidor un fichero que defina el formato.
- **Políticas de retención**: Nombre de la política.
- **Tipo de documento**: Descripción y tipo de documento.
- **Parámetros globales**: Nombre del parámetro.
- **Instituciones**: Identificador del emisor y nombre del emisor.
- **Sectores**: Identificador del sector y descripción.

- **Contrataciones**: Identificador del sector, identificador de la institución emisora, tipo de formato y si está activa o no.

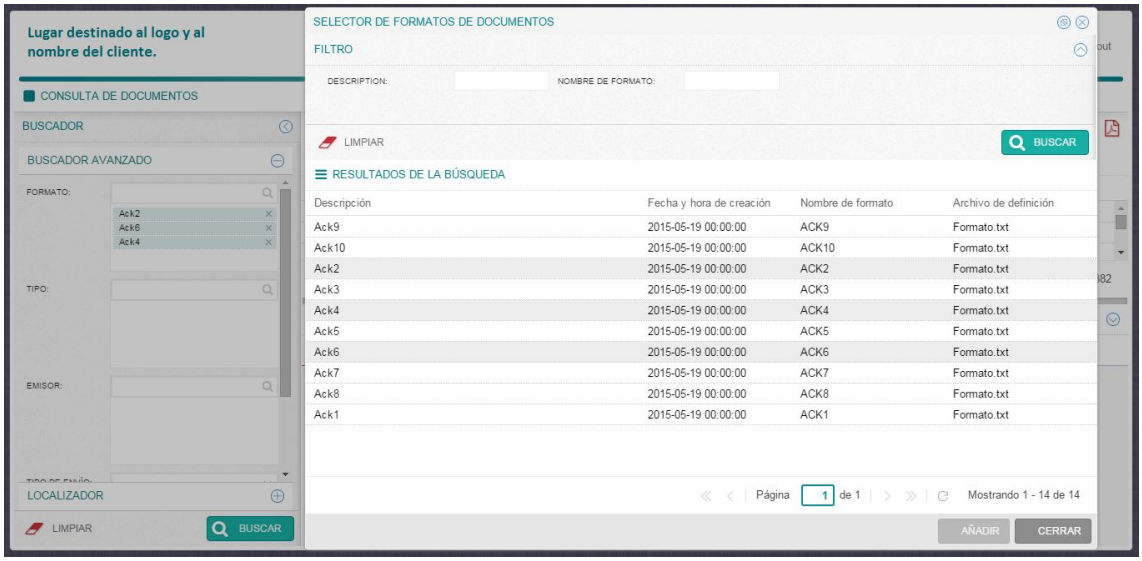
Desarrollo Front-End

En estas siete pantallas se ha seguido un mismo procedimiento: colocar el buscador en la parte norte de la vista maestra, y la lista en la parte central. La vista detalle de cada mantenimiento se compone de un formulario con los campos necesarios para dar una nueva alta o modificar un registros. Para ambas acciones los parámetros son los mismos. Hasta que el formulario no es válido, el botón de guardar no se activa.

Para algunos campos pertenecientes a las búsquedas y al formulario de la vista detalle, se ha diseñado un nuevo componente, ya que EXT JS5 no lo proporciona por defecto. El nuevo elemento funciona como un combo, en el que en lugar de mostrar una lista de posibles opciones, aparece una ventana con un buscador y una tabla de resultados. Al pulsar sobre uno de ellos, la ventana se cerrará y dará ese valor al campo. A este componente se le ha llamado *buscador*.

Cuando se programa una vista en la que se inserta el *buscador*, se le puede dar el atributo "*multiselect: true*". Con esto, se consigue que al aparecer la ventana de búsqueda, se permita seleccionar varios resultados. En la aplicación solamente se ha proporcionado esta configuración cuando el componente forma parte del buscador de la vista detalle. Un ejemplo visual de un *buscador* en el que se permite seleccionar varios elementos de la lista, los cuales se insertan automáticamente en el nuevo componente, es:

Ejemplo de la ventana del buscador múltiple de formatos de documentos.

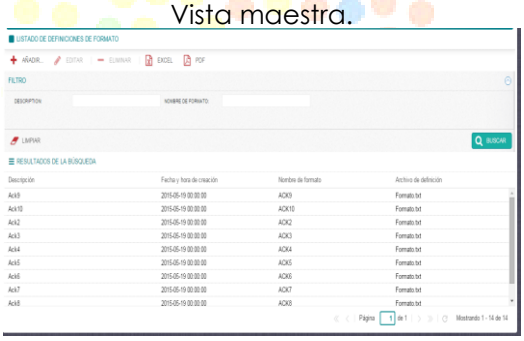


[Imagen 41]

Visualización

Formato de documento.

Vista maestra.



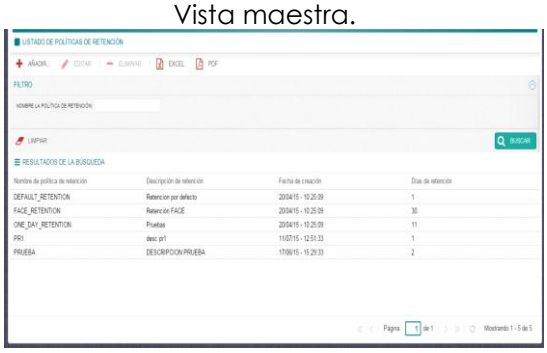
[Imagen 42]

Vista detalle.

[Imagen 43]

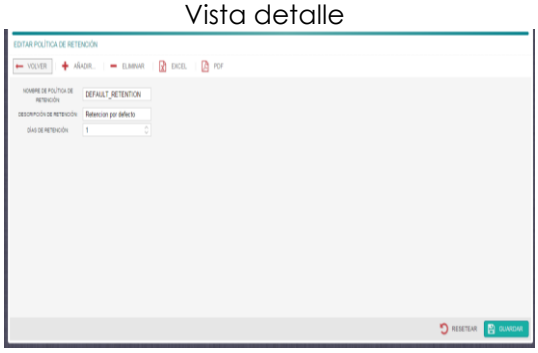
Políticas de retención.

Vista maestra.



[Imagen 44]

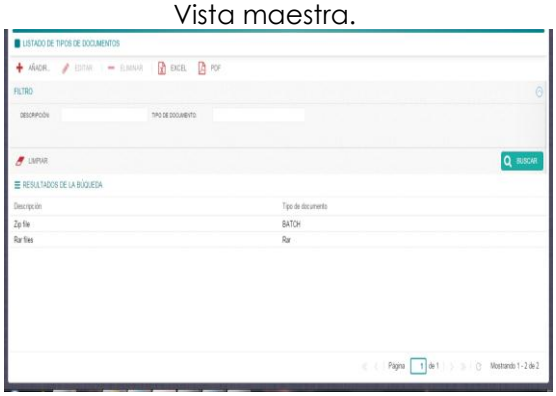
Vista detalle



[Imagen 45]

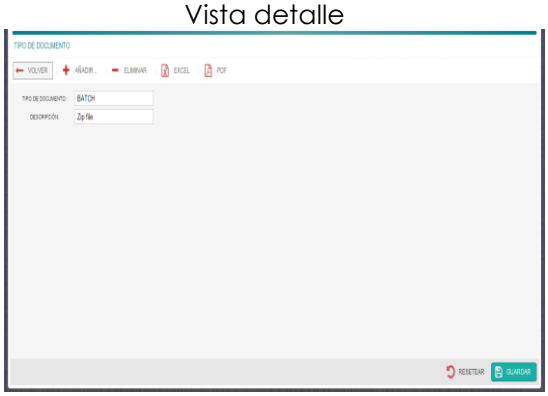
Tipo de documento.

Vista maestra.



[Imagen 46]

Vista detalle



[Imagen 47]

Parámetros globales.

Vista maestra.

[Imagen 48]

Vista detalle

[Imagen 49]

Instituciones.

Vista maestra.

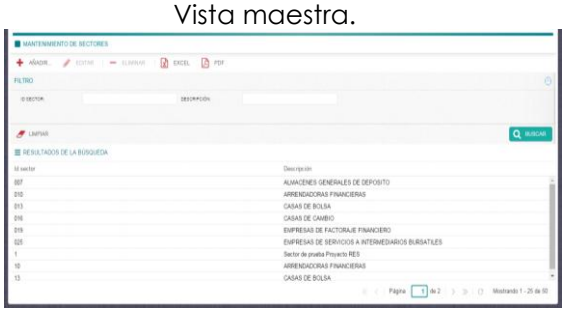
[Imagen 50]

Vista detalle

[Imagen 51]

Sectores.

Vista maestra.



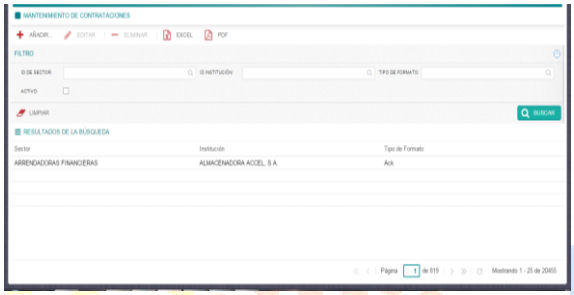
[Imagen 52]

Vista detalle

[Imagen 53]

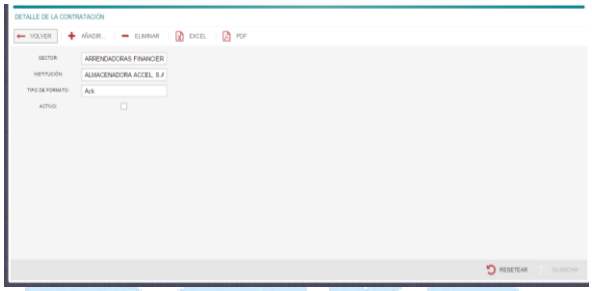
Contrataciones.

Vista maestra.



[Imagen 54]

Vista detalle



[Imagen 55]

5.9. ADMINISTRACIÓN DE MANTENIMIENTOS COMPLEJOS

Especificaciones del cliente

Para los mantenimientos de instalaciones, workflows y reglas de procesamiento, las vistas son más complejas, ya que deben mostrar relaciones "1 a n".

En **instalaciones** se debe filtrar la lista de resultados por identificador y por descripción. Cada instalación tiene asociadas varias instituciones y como cada una de éstas puede tener vinculadas diferentes contrataciones activas, también deben ser visualizadas. Se podrán crear nuevas instalaciones o modificarlas. También se podrán asociar nuevas instituciones o modificar las ya existentes. Debido a que las contrataciones llevan asociadas el identificador de la institución a la que pertenecen, las altas o modificaciones de estos registros solamente deberán realizarse en su propio mantenimiento.

En la pantalla de **workflows** se debe visualizar un buscador que filtre por: nombre, formatos de documentos, tipos de documentos, instituciones emisoras y referencia de la aplicación. Al *workflow* se le

vincularán diferentes etapas, las cuales podrán tener asociadas diferentes reglas, creadas en el mantenimiento de reglas de procesamiento. Por lo tanto, desde esta pantalla se podrán crear y modificar tanto los workflows como las etapas, mientras que para las reglas, solo se podrán asociar o desasociar.

En **reglas de procesamiento** solamente habrá una relación "1 a n". Ésto se debe a que cada regla puede constar de varios parámetros. Por lo tanto, se crearán y modificarán las reglas de procesamiento a las que se podrán asociar diferentes parámetros. El buscador debe filtrar por: nombre de regla, tipo de regla, instituciones emisoras, referencia de la aplicación, tipos de documentos y formatos de documentos.

Desarrollo Front-End

Para estas tres pantallas, se ha seguido una maquetación similar. Se ha colocado el buscador a la izquierda, con lo que se consigue que en la parte central haya más espacio para visualizar las diferentes tablas.

Cuando se pulsa sobre uno de los registros, se recarga el grid de la relación que tiene asociada directamente; y si hay relaciones indirectas, el grid en las que se visualizan, se limpia. Para cargar las relaciones de un modelo, se realiza una búsqueda enviando como filtro el identificador del registro. Al crear una nueva vinculación o modificar una existente, se envían a la base de datos tanto los valores introducidos por el usuario, como el identificador del registro al que se debe asociar. Cuando un registro es borrado por el usuario, se limpian los grid de las relaciones que tenga. El Back-End se encarga realmente de borrarlas de la base de datos aunque no haya recibido un método DELETE sobre los registros secundarios que se deben eliminar.

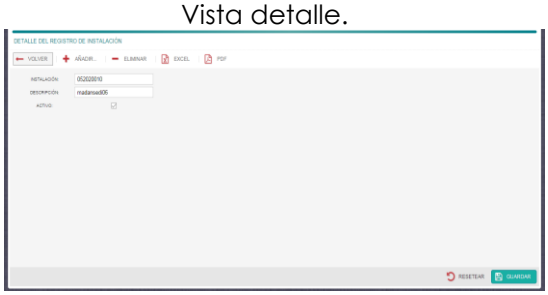
Visualización

Instalaciones.

Vista maestra.

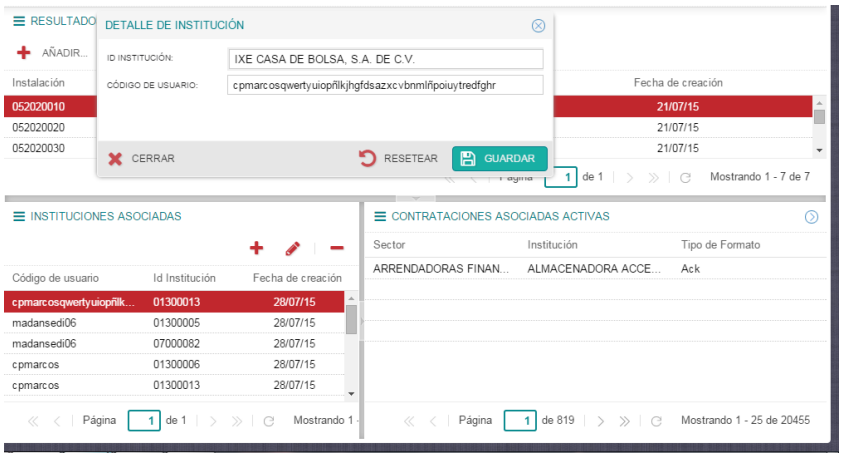
[Imagen 56]

Vista detalle.



[Imagen 57]

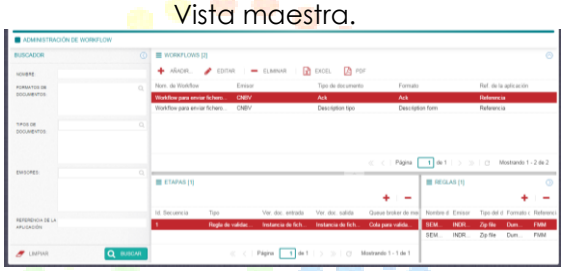
Detalle de institución asociada a la instalación.



[Imagen 58]

Workflows.

Vista maestra.

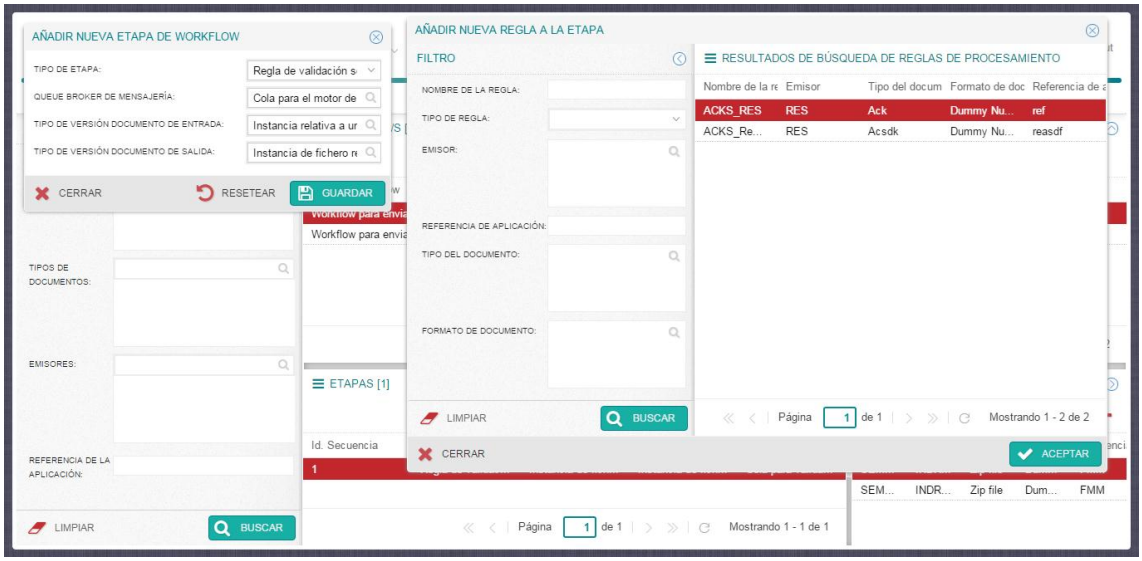


[Imagen 59]

Vista detalle.

[Imagen 60]

Ventanas emergentes para añadir una nueva etapa al workflow y asociar una reglas existente a la etapa.



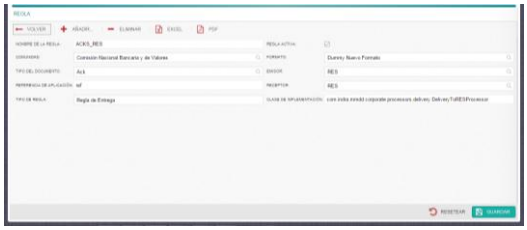
[Imagen 61]

Reglas de procesamiento.

Vista maestra.

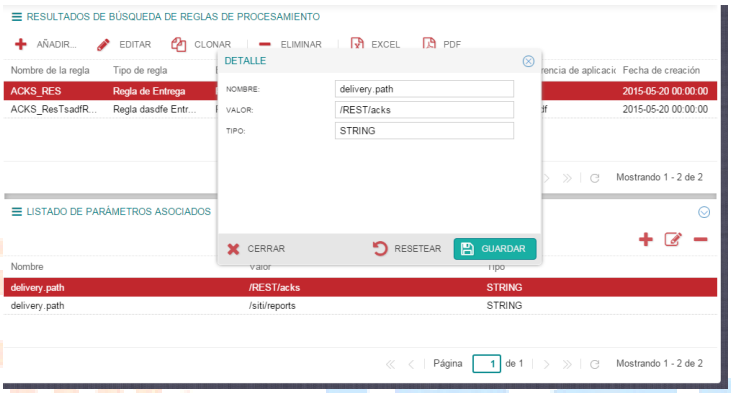
[Imagen 62]

Vista detalle.



[Imagen 63]

Detalle del parámetro asociado.



[Imagen 64]

5.10. CALENDARIOS INHÁBILES

Especificaciones del cliente

Se precisa de una pantalla en la que se muestren los diferentes días que no son hábiles para el cliente. Se podrán añadir más días o eliminar los ya marcados.

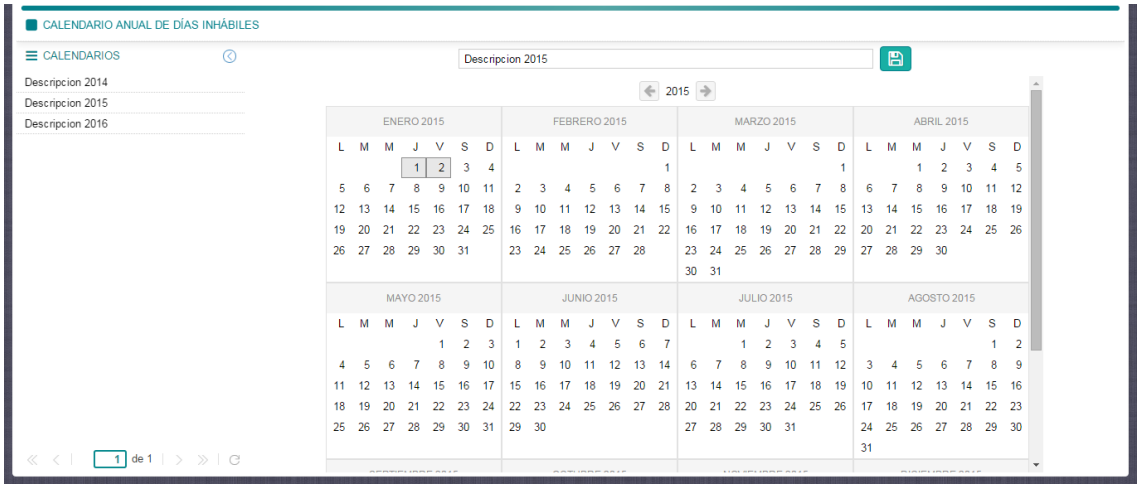
Desarrollo Front-End

En la parte izquierda de la pantalla se muestra una lista con los diferentes años en los que se ha indicado un día festivo. En la parte central se visualizan todos los meses del año seleccionado en la lista, mostrándose en cada uno de ellos los días de inhabilidad. Para añadir un nuevo día, simplemente se selecciona; para eliminarlo, bastaría con deseleccionarle.

Si se desea añadir un día festivo en un año que no aparece en la lista, se deberá localizar el año y añadir una descripción. Una vez hecho esto, ya se podrán marcar los diferentes días.

Visualización

Vista de calendarios inhábiles



[Imagen 65]

5.11. CUADRO DE MANDOS

Especificaciones del cliente

El cliente desea poder visualizar una serie de gráficas en las que se realizan diferentes seguimientos. En las gráficas se mostrará el número de documentos enviados con y sin errores, el número de documentos totales enviados por horas, el número de documentos enviados con y sin errores por institución y el número de envíos correspondientes a cada institución. También se desea visualizar una lista con las instituciones que no han emitido documentos. En todas las búsquedas, se debe indicar el periodo en el cual se desea obtener los resultados.

Desarrollo Front-End

La pantalla de cuadro de mandos, es la más compleja que tiene la aplicación. Lo primero para que EXT JS5 pueda trabajar con gráficas, es que los ficheros correspondientes al *sencha-charts* estén ubicados dentro del archivo *app.json*.

La vista se compone de: un buscador que parece muy simple visualmente, pero es bastante complicado de programar; de una lista con los resultados de la búsqueda, y del gráfico que los datos han generado.

El buscador está inicialmente formado por un *segmentedbutton*. Dependiendo del botón al que se pulse, se mostrarán unos ítems distintos.

Cada botón indica un tipo de cuadro de mando diferente. Las características son:

1. **Número de Ok vs Ko**: Al seleccionar este botón, se muestran dos combos, el primero para seleccionar la agregación y el segundo para marcar el tipo de valor que se desea que se muestre en las gráficas (absoluto o relativo). Según el tipo de agregación que se ha seleccionado, variará tanto el buscador como los datos que aparecen en el eje x de la gráfica. En el gráfico de barras generado en este caso, se visualizan los documentos correctos contra los incorrectos.

   Los diferentes tipos de agregación son:

   a. **Día actual**: Se obtienen los datos de una tabla de la base de datos, que almacena todos los envíos que se han dado desde las 00:00 horas hasta el momento actual.

   b. **Diario**: Al finalizar el día, se almacenan en la base de datos, la fecha y el número de transmisiones correctas con fallos y totales que se han generado para ese día. El periodo del cual se quieren obtener los datos, está limitado a un mes, ya que al ser la agregación diaria, el número máximo de barras que contiene la gráfica es treintaiuno.

   c. **Mensual**: En este caso, se puede introducir desde qué día a qué día se quieren visualizar los resultados. Al ser la agregación anual, el número máximo de barras que tiene la gráfica es doce, ya que la consulta a la base de datos, devuelve el resultado de las operaciones que se producen al sumar el número de transiciones (por un lado de documentos enviados sin errores y por otro de enviados con errores) que se dan por meses, en el periodo seleccionado. Se facilita al usuario la opción de introducir en el apartado cerrado del selector de periodo, el año de los meses que quiera visualizar.

   d. **Anual**: Es similar a mensual, pero en este caso, el resultado de las operaciones será el correspondiente a

la suma de transiciones erróneas y correctas que se dan por año. El número máximo de años que se visualizan es cinco. Se facilita al usuario la opción de introducir en el apartado cerrado del selector de periodo, los cinco años cuyas transacciones le interesen.

e. **Instituciones**: En esta opción aparecen nuevos filtros: un tercer combo, en el que se obliga a seleccionar si el usuario quiere observar únicamente el número de documentos correctos o erróneos que ha generado una institución, y cuál es el número de instituciones que quiere que se visualicen en la gráfica (veinticinco máximo). En este caso, los datos pertenecerán a un periodo concreto, ya sea mes, trimestre, semestre o año, desde la fecha que el usuario introduce. Si en el combo de tipo valor se selecciona relativo, se muestra el porcentaje de los correctos contra los erróneos de una determinada institución.

2. **Totales**: En este botón se da unicamente la opción de seleccionar el tipo de agregación. En este caso serán:

   a. **Horas**: Una vez seleccionado el periodo del que se quiere obtener los datos (máximo un año), se visualiza en una gráfica, de puntos y líneas, el número de envíos que se han dado en una hora concreta a lo largo de todo el periodo.

   b. **Instituciones**: El periodo para el que se quiere realizar la observación, funciona de la misma forma que en la agregación por instituciones en número de Ok vs Ko. En este caso, en la gráfica de puntos y líneas se muestra el tamaño de los envíos que ha generado cada institución. Nuevamente se podrá elegir el número de instituciones que se quieren mostrar, siendo también veinticinco el número máximo.

3. **Sin envíos**: En este botón no se muestra ninguna gráfica, ya que la búsqueda solo genera una lista de resultados, con todas las instituciones que no han generado transiciones para un determinado periodo. Como es una lista de instituciones, el periodo es el mismo que en los dos casos anteriores de agregación por instituciones.

Cuando se realiza una búsqueda que tiene relación con instituciones, se puede filtrar, además de lo ya explicado, por sector;

mientras que si no lo está, también se puede filtrar por institución, sector y formato.

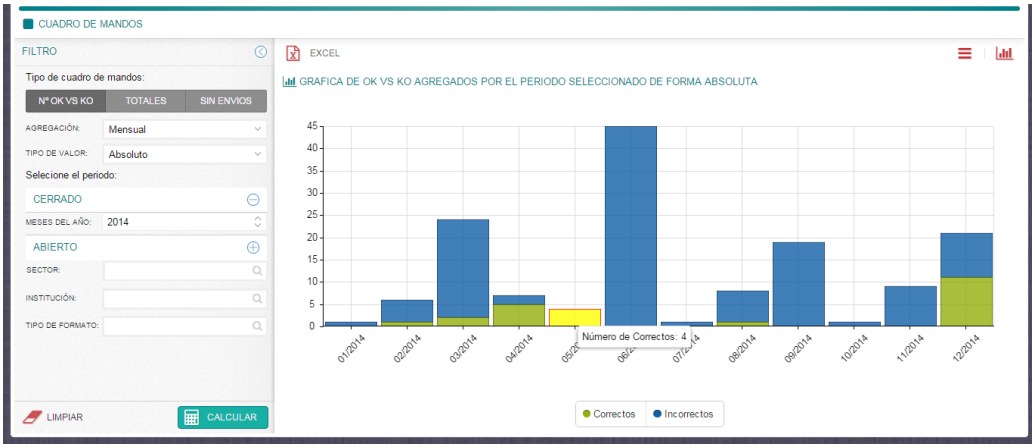
La complicación de esta pantalla está en cómo realizar la interacción entre los diferentes componentes, y cómo conseguir que dependiendo de los botones pulsados, se realice una petición a un servicio o a otro. El procedimiento que se ha seguido para cumplir el objetivo, ha sido el de diseñar una máquina de estados en el controlador de la vista, que realmente es un array con el nombre del botón del *segmentedbutton* en la primera posición, y el del tipo de agregación en la segunda.

Cuando se pulsa el botón "Calcular", se observa la máquina de estados, y dependiendo de los valores que tenga, se reconfiguran las columnas que deben aparecer en la tabla, y se selecciona el *store* concreto que se necesita, y éste llama al método *load()*. Con esto, se consigue realizar la petición al recurso, que está definido en la url del proxy del modelo que sigue el *store*.

Una vez que el *store* posee datos, se inserta en la tabla. Si la máquina de estados indica que se ha hecho una búsqueda en la que se debe mostrar una gráfica (siempre que en la primera posición del array no haya el *string* "sin envíos"), se observan los ítems existentes en el contenedor de la parte central de la pantalla. Acto seguido, se borra el último (ya que será el panel del gráfico anterior) y se inserta el panel que contiene la nueva gráfica, a la cual se le carga el *store* que acaba de ser rellenado con los datos provenientes de la consulta. En el caso de que no haya gráfica, simplemente se deshabilita el botón que la muestra.

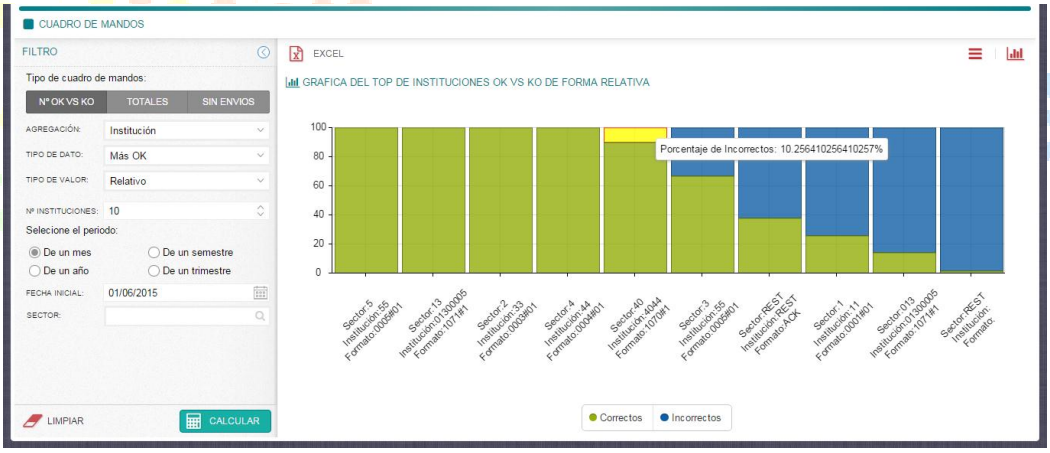
Visualización

Buscador y gráfica de número de envíos Ok vs Ko agregados mensualmente de forma absoluta. El periodo seleccionado es el del año 2014 completo.



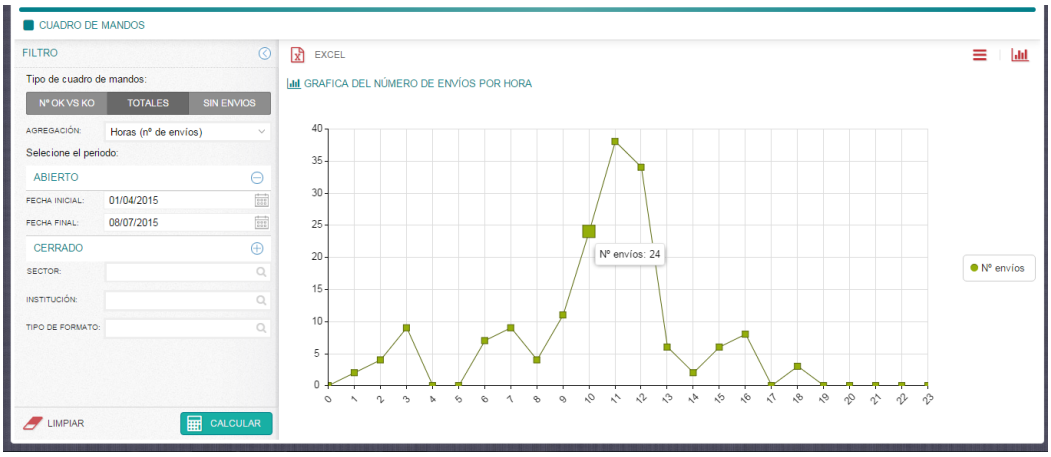
[Imagen 66]

Buscador y gráfica de instituciones con más documentos enviados correctamente en un mes desde el 01/06/2015 de forma relativa.



[Imagen 67]

Buscador y gráfica del número de envíos por horas desde el 01/04/2015 hasta el 08/07/2015.



[Imagen 68]

Buscador y lista de instituciones que no han enviado documentos en un trimestre desde el 23/10/2014.

[Imagen 69]

6. PRESUPUESTO, PLANIFICACIÓN Y RESTRICCIONES

En este apartado, se describe el presupuesto que se puso a disposición del cliente cuando se interesó por la empresa para la realización de su aplicación web. También aparece el Diagrama de Gantt del proyecto, en el que se describen las diferentes actividades y los tiempos que han durado. Para concluir este capítulo se detallarán las regulaciones que se han tenido que tener en cuenta.

6.1. PRESUPUESTO

Para realizar el presupuesto de un proyecto real, se tienen que tener en cuenta todas las personas que trabajan en él. En este caso, han formado parte del equipo: un jefe de proyecto, un comercial, un analista, cuatro programadores de Front-End (un ingeniero de sistemas experto, uno senior, uno básico y un becario (autor del proyecto)) y tres programadores de Back-End (un ingenieros de sistemas experto y dos básicos). Éstos últimos también son los encargados de realizar las diferentes pruebas a la aplicación.

6.1.1. COSTES PERSONALES

Estos costes están asociados tanto al número de personas que trabajan en el proyecto, como al número de horas que le han dedicado. Dependiendo del rol que tiene cada trabajador, el coste que supone al cliente es diferente (el coste y el número de horas que han dedicado los desarrolladores han sido ocultados por petición expresa de la empresa.)

Tabla con los costes personales detallados.

Actividad	Rol	Coste (por horas)	Número de horas	Coste total
Vender el producto al cliente	Comercial	*	*	*
Redactar el funcional de la aplicación	Analista	*	*	*
Programar el Back-End	Ingeniero de sistemas experto	*	*	*
Programar el Back-End	Ingeniero de sistemas	*	*	*
Programar el Back-End	Ingeniero de sistemas	*	*	*
Diseñar estilos para el Front-End	Ingeniero de sistemas experto	*	*	*
Programar el Front-End	Ingeniero de sistemas senior	*	*	*
Programar el Front-End	Ingeniero de sistemas	*	*	*

Programar el Front-End	Becario (Autor de la memoria)	*	*	*
Realizar test	Ingeniero de sistemas experto	*	*	*
Resolver incidencias en Back-End	Ingeniero de sistemas	*	*	*
Resolver incidencias en Front-End	Becario	*	*	*
Elaborar la memoria del proyecto	Analista	*	*	*
Revisión	Jefe de proyecto	*	*	*
TOTAL				*

[Tabla 2]

6.1.2. COSTES DE HARDWARE

Los costes de hardware son generados por los materiales físicos que se han utilizado al realizar el proyecto. Los diferentes elementos no se compran nuevos cada vez que se inicia un proyecto, sino que se reutilizan, por lo que su tiempo de "vida" no se limita a la duración del proyecto. A pesar de ello, cada cierto tiempo se deben renovar (cuando se cumple su periodo de depreciación), por lo que únicamente se cobra un porcentaje del coste total del hardware al cliente.

Desglose del coste de hardware.

Concepto	Cantidad	Coste sin IVA	Periodo de depreciación	Coste imputable	Coste total
Ordenador HP i5	10	499,99 €	5 años	10%	499,99 €
Ratón HP Óptico	10	19,99 €	2 años	20%	39,98 €
Monitor 19"	7	120,00 €	7 años	5%	42,00 €
Servidor Microsoft	1	2.000,00 €	10 años	25%	500,00 €
TOTAL					**1.081,97 €**

[Tabla 3]

6.1.3. COSTES DE SOFTWARE

Los costes de software son producidos debido a que los programadores necesitan tanto librerías o frameworks, como programas de compilación, para realizar una aplicación web. Algunos son gratuitos, pero otros son de pago. Las ventajas de comprar licencias son:

- Los programadores no tienen que picar todo el código desde cero, si no que pueden utilizar una gran cantidad de métodos ya definidos.

- Las empresas que han diseñado los frameworks dan soporte a los programadores, esto quiere decir que si éstos últimos detectan cualquier tipo de fallo durante el desarrollo de la aplicación debido a un error en la librería, podrán ponerse

en contacto con la empresa desarrolladora para que lo arreglen.

Los costes que suponen los diferentes softwares utilizados en la aplicación son:

Concepto	Cantidad	Coste	Coste total
Licencia Ext JS5	1	7.580,09 €	7.580,09 €
Licencia WebStorm	2	48,00 €	96,00 €
Licencia Subime Text	1	0,00 €	0,00 €
Licencia Eclipse	2	0,00 €	0,00 €
Licencia Spring	2	0,00 €	0,00 €
Licencia Microsoft SQL Business Intelligence	1	8.974,62 €	8.974,62 €
TOTAL			**16.650,71 €**

[Tabla 4]

6.1.4. COSTES TOTALES

El coste sin IVA es el generado al sumar los costes de personal, hardware y software. A este valor, se le debe añadir otros costes para calcular el precio total, que el cliente debe pagar para que el proyecto se lleve a cabo (los porcentajes y el precio final han sido ocultados por petición expresa de la empresa):

1. El margen de beneficios que obtiene la empresa al desarrollar el proyecto. Para este caso se ha considerado un x % respecto al coste sin IVA.

2. Los costes generados por el margen de riesgo se deben a que si el proyecto se retrasa por alguna causa, o hay algún contratiempo, la empresa puede disponer de unos ingresos que utiliza para solventar estas dificultades. Como los programadores utilizan frameworks que ya conocen, el porcentaje del margen de riesgos es únicamente del x%. Este coste se calcula respecto a la suma de los costes sin IVA y del beneficio que la empresa obtiene.

3. El último coste que se le añade al proyecto es el del impuesto sobre el valor añadido (IVA). Actualmente en España este porcentaje es del 21%. Este valor se aplica a la suma de los costes calculados hasta el momento.

Finalmente, el presupuesto que se pone a disposición del cliente le ocasiona un gasto de X€ .

La tabla con los costes finales.

Concepto	Coste
Coste personal	*
Coste de hardware	1.081,97 €
Coste de software	16.650,71 €
Coste sin IVA	*
Beneficios	*
Margen de riesgo	*
IVA (21%)	*
COSTE TOTAL	*

[Tabla 5]

6.2. PLANIFICACIÓN: DIAGRAMA DE GANTT

Una vez que el cliente acepta el presupuesto que el comercial le ha ofrecido, el analista gestiona el tiempo que debe ocupar cada actividad que alberga el proyecto. A continuación se muestra el detalle de las actividades e hitos que se tienen que completar en el Front-End para que éste cumpla los objetivos que se le han designado.

En la columna designada al nombre, se puede observar distintos niveles de identación, que se utiliza para marcar la estructura de las tareas del proyecto. Para que una tarea padre esté completa, tienen que estarlo todas sus hijas. También se puede observar que hay nombres en negrita. Éstos corresponden al nombre de la actividad. Si el formato de texto es normal, significa que es un hito del proyecto.

Tabla de actividades realizadas durante el desarrollo del Front-End de la aplicación web.

		Nombre	Duracion	Inicio	Terminado	Predecesores
1		**Front-End de la aplicación**	**116 days**	**6/04/15 8:00**	**14/09/15 17:00**	
2		**Estructura del proyecto**	**5 days**	**6/04/15 8:00**	**10/04/15 17:00**	
3		Crear Front-End	1 day	6/04/15 8:00	6/04/15 17:00	
4		Diseñar pantallas en PhotoShop	4 days	6/04/15 8:00	9/04/15 17:00	
5		Definir temas y estilos para la aplicación	1 day	10/04/15 8:00	10/04/15 17:00	4
6		**Maquetación de pantallas**	**106 days**	**13/04/15 8:00**	**7/09/15 17:00**	**2**
7		Jsp del Login	2 days	13/04/15 8:00	14/04/15 17:00	
8		Menú	3 days	15/04/15 8:00	17/04/15 17:00	7
9		Cambio de datos del usuario logueado	6 days	20/04/15 8:00	27/04/15 17:00	8
10		Mantenimiento de usuarios	3 days	20/04/15 8:00	22/04/15 17:00	8
11		Auditoria de documentos	5 days	28/04/15 8:00	4/05/15 17:00	9
12		Audiotoria de transmisiones	3 days	23/04/15 8:00	27/04/15 17:00	10
13		Mantenimiento de formatos de documentos	3 days	28/04/15 8:00	30/04/15 17:00	12
14		Mantenimiento de políticas de retención	5 days	5/05/15 8:00	11/05/15 17:00	11
15		Mantenimiento de tipos de documentos	2 days	1/05/15 8:00	4/05/15 17:00	13
16		Mantenimiento de parámetros globales	4 days	12/05/15 8:00	15/05/15 17:00	14
17		Mantenimiento de instituciones emisoras	4 days	18/05/15 8:00	21/05/15 17:00	16
18		Mantenimiento de calendario de días inhábiles	5 days	5/05/15 8:00	11/05/15 17:00	15
19		Mantenimiento de sectores	4 days	22/05/15 8:00	27/05/15 17:00	17
20		Mantenimiento de contrataciones	4 days	28/05/15 8:00	2/06/15 17:00	19
21		Mantenimiento de intalaciones	10 days	3/06/15 8:00	16/06/15 17:00	20
22		Mantenimiento de workflows	10 days	17/06/15 8:00	30/06/15 17:00	21
23		Mantenimiento de reglas de procesamiento	9 days	1/07/15 8:00	13/07/15 17:00	22
24		Cuadro de mandos	20 days	14/07/15 8:00	10/08/15 17:00	23
25		Diseñar componente buscador	8 days	12/05/15 8:00	21/05/15 17:00	18
26		Administración del cierre de sesión	15 days	11/08/15 8:00	31/08/15 17:00	24
27		Tratamiento de los códigos de error en las respuestas del servidor	5 days	1/09/15 8:00	7/09/15 17:00	26
28		Resolver incidencia	5 days	8/09/15 8:00	14/09/15 17:00	27

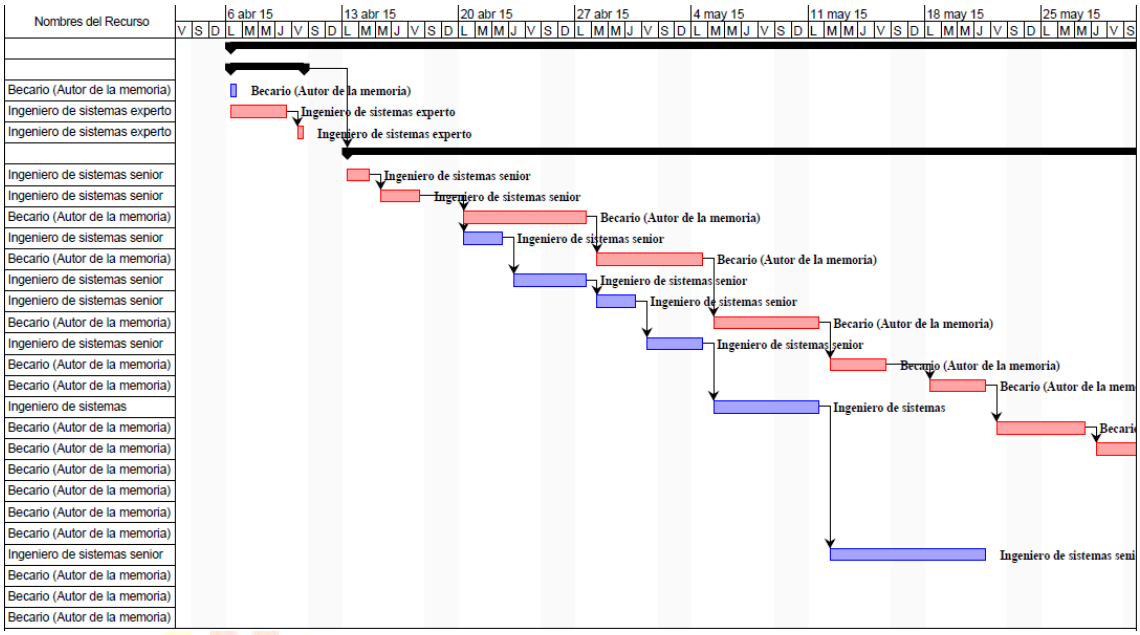
[Tabla 6]

En el diagrama de Grantt se representa el tiempo que el analista ha estimado que dura cada actividad e hito. En él se puede distinguir tres colores diferentes para los marcadores del tiempo:

- Negro: representa el periodo que dura una actividad.
- Rojo: representa al periodo de un hito crítico en el tiempo. Si éste es superado, las actividades siguientes comenzarán retrasadas, por lo que el final del proyecto se alargará.
- Azul: representa el periodo de duración de un hito no crítico en el tiempo.

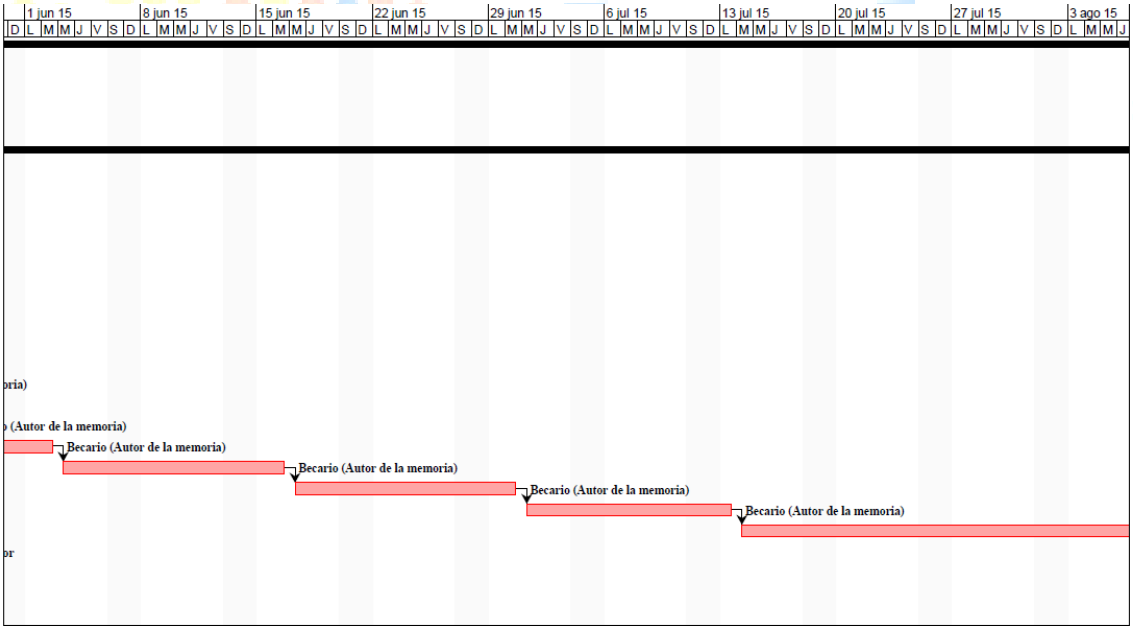
En la columna llamada nombre del recurso se asigna al rol de la persona encargada ese hito. Esto no quiere decir que solamente ese programador trabaje en el hito, ya que los diferentes componentes del equipo, si ha necesitado el encargado ayuda, han colaborado abiertamente.

Diagrama de Gantt desde el 06/04/2015 al 30/05/2015.



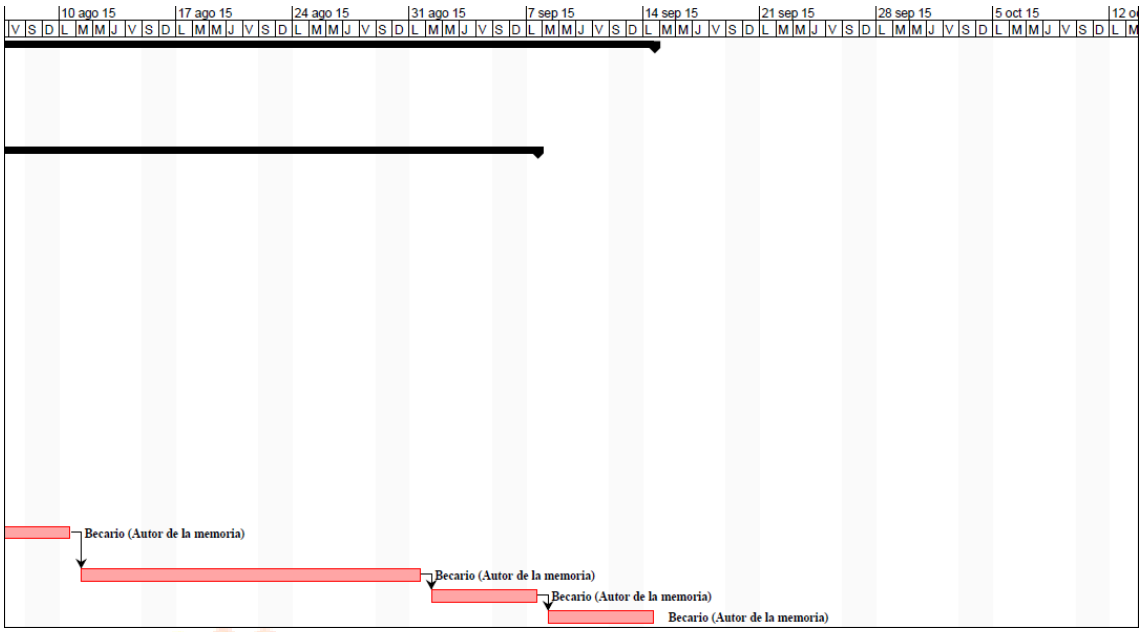
[Diagrama de Gantt parte 1]

Diagrama de Gantt desde el 31/05/2015 al 06/08/2015.



[Diagrama de Gantt parte 2]

Diagrama de Gantt desde el 07/08/2015 al 14/09/2015.



[Diagrama de Gantt parte 3]

Finalmente, se quiere explicar que antes de dar por cerrado un hito, el Front-End y el Back-End se tienen que poner en contacto para comprobar la integración entre cliente-servidor. Si se encuentran fallos, será el momento de solucionarlo. Este periodo también está incluido en la duración del hito dentro del diagrama.

6.3. RESTRICCIONES EN LA APLICACIÓN

En el proyecto que se ha llevado a cabo, no ha habido un organismo regulador específico, si no que se han seguido varias especificaciones como son las peticiones que el cliente ha demandado y las directrices que el jefe del proyecto y los programadores expertos han marcado.

Además de esto, se han seguido las directrices que toda aplicación CRUD (Create Read Update Delete) tiene. Como el estilo de la arquitectura se ha apoyado fundamentalmente en el estándar de HTTP (los servicios de comunicación entre cliente y servidor son de tipo RESTful). Esto significa que:

- Para guardar datos en la base de datos se utiliza el método POST, para leer registros el GET, para modificarlos el PUT y para borrarlos el DELETE.
- Los códigos que tienen asociadas las respuestas enviadas por el servidor son los mismo que aparecen en el estándar de HTTP:

RFC 2616; y en algunos fragmentos de los estándares: RFC 2518, RCF 2817, RFC 2774, RFC 4918.

- Las URIs siguen la forma de directorios.
- Las comunicaciones entre cliente y servidor son mediante ficheros JSON.

También se puede considerar como restricción de la aplicación a los acuerdos que se tienen que realizar entre los programadores de Front-End y de Back-End para que la comunicación entre ellos sea exitosa.

7. SUMMARY (ENGLISH)

Summary of this project report will be detailed next.

The End of Degree Project consists of developing a web application with one of the most powerful market's frameworks: EXT JS. Author would like to point that this framework is based on JavaScript language. Most companies ask for programmers that know how to program this language because of numerous functionalities that offers.

There are many frameworks available in the current labor market (such as JQuery, Node JS, Angular…) which use JavaScript. It has been decided to use EXT JS 5 for its numerous advantages:

- It has got a lot of documentations where it is possible to visualize code fragments which can be executed and modified in order to show the changes that this causes.
- There is a perfect compatibility between different components that Ext JS developers have implemented.
- The web application appearance is programmed in JavaScript and the developer does not need to use HTML. When the browser downloads and executes JavaScript files, the EXT JS framework internally calls to a method that inserts different labels in the web page´s body. These vary depending on the component that the programmer has implemented in the JavaScript file. The web page´s head is always the same because the browser captures it from the index.html.
- There are many ways of placing the numerous elements that are showed in a view. This way of organization is easily do with this framework. For example: if a developer wants to place four elements in 2x2, he would only have to specify that he wants a layout of column type. He must program that each column takes exactly the 50% of it. As the two first elements take up all space of the line, last elements are showed in a line below. If all this was in HTML programmed, this would have been more difficult. In HTML, the programmer had had to define a table with two columns and two rows and he had had to insert one element in each one.
- The user can change the side of emerging windows without the developer programing this functionality. If this is not the expected performance, the programmer can give the value

false to the "resizable" attribute component. Other performances that can be changed by default are: Organizing columns in grids, changing the columns width in grids, showing scroll arrow every time that needed it…

- A component can be programed for executing a method when the user makes an action. Also the element can listen to an event and then it executes another function.
- The EXT JS framework perfectly implements the RESTful software architecture, which all CRUD web application must follow.

Not all are advantages from this framework, EXT JS has disadvantages too, as the following ones:

- The learning of this framework is quite hard if the programmer has not implemented any program in JavaScript.
- As there are a lot of components in EXT JS 5, the developer needs much more time in knowing all of them.
- The waiting time to initialize an EXT JS web application is greater than the time need it with any application created in other framework. This is because the browser needs to download all JavaScript file that EXT JS has. In doing so, the application will operate correctly.

After what it has been said above, the author recommends using EXT JS only in web application that replaced old desktop programs; while in normal web pages where not needed powerful components, as the others, and where the user waiting time will be reduced in length.

The EXT JS developer must know different elements that form the framework. These are organized in the next way:

- Model: Is an element that it has data that came from a database query. The developer has to define data that the model can control in order to give them different formats, and also, to change their representation in the view. He must program the proxy too; the proxy type (the type of software architecture that follows client-server communication). And the developer must also program the URL where to ask for the request. The software architecture used normally is RESTful. This means that the client-server communication uses the following items: HTTP methods (GET, POST, PUT and DELETE), errors codes (which are the same that HTTP responses), URIs that follow the

directory form, and finally, communication between the client and the server is than done by JSON files. Having said that, to replenish the data model it is necessary to specify the identifier resource which holds the database. When the method "load(<identifier>)" is called by the model, it generates automatically the following URI: "service/identifier" and it fulfills the data model. The HTTP method is GET in the above case. If the method "save()" is called by phantom model (it has no identifier in the database), a new element will be created in the database, but if not, it will modify the old element. The different between both "save()" method is that the first one executes POST HTTP method and its URI is: "service". While the other one, the HTTP request executes the PUT method instead and URI is "service/identifier". Finally, if the model calls to "erase()" function, the resource will be deleted in data base. For this last one, the URI is "service/identifier" and HTTP method is DELETE.

- Store: It is an object that stores data models of same types. If the store calls the "load()" method, all elements will be loaded from service. This service is accessible through the model URI that contains the store. The HTTP method is GET and the URI is: "service". The different with the "load()" method model is that an identifier has not been specified now, that is the reason why all service resources are sent to the service response.
- View: Here are defined the components that are showed in a specify view. These can be: containers and panels (these are used to contain other elements). Other are: buttons, textfields, textareas, grids, checks, datefields, combos... The difference between a container and a panel is that the panel can have a title and a toolbars. Components that contain other elements have a special attribute called layout. This is used for placing the components that it contains.
- Viewcontroller: In these files are methods that are needed to execute when the user interacts with components of a particular view. Some examples of actions that the user can make in a view are: to press a button, to write in a textfield, to change the value of a combo, to select a date in a calendar, to select a row in a grid, to make double click in a grid element… Components of a view can also fire events or listen them. Methods that these actions execute are in the viewcontroller too. Finally, requests that are sent to server by

client are met here too. From here it is possible to program functions that the developer wants to execute when a response is received or when is not successfully done. The programmer can also implement the callback request. Each viewcontroller must be defined for an only view.

- Viewmodel: The developer implements a different viewmodel for each view. In this file is programed all relations between the model and the view. "Databinding" got this, for example: To connect a textfield with a model data, so that if a user writes in that textfield, automatically the model data will be replenished.

It is possible to change the display style of an EXT JS web application by means of two procedures:

- To create css files and to give different styles according to the traditional form.
- And to develop a characteristic theme for the application. This is done thanks to the sass program language. To compile these files it is necessary the compiler "Composer". If the developer uses the program designed by Sencha, "Sencha CMD", he will add "Composer" and also a web server to execute the application. Theme is defined with the purpose that all equal components have the same styles. Styles would not be applied if sass file routes do not follow the same structure that Sencha uses to program their components. The theme files are located in the directory package. To give specific styles to a unique view, sass files are used too; but in this last case they are located in the directory application folder sass. To show styles, these sass files must follow the structure that view files have.

The author of this End of Degree Project has developed a web application example named "Hola Fifa". It is possible to download it through the GitHub author: https://github.com/thebohodon/HolaFifa-EXTJS-5. The application has two parts: a server and a client. To execute it, the interested one must install before Node JS. This is why the server is programmed with it. Thanks to this, all modifications that the user can make in the client part (to create a football player, to modify it or to delete it), which are programmed with EXT JS 5, are saved in JSON files.

The demo application had a problem because the application is implemented with the Node JS server. As the author has not used the "Sencha CMD" program, he could not create an application theme and

not to give styles by sass means. The component styles were given thanks to a css file.

The End of Degree Project that has been developed is about the Front-End web application. In this, banking documents transmissions are monitored and to check their details is also possible. As the project has been developed in Indra Company, client data will not be revealed nor the web application real price either.

Before beginning to program the Front-End, it is highly recommended to design all application views either in a paper or in Photoshop. In addition if project files are organized tidely, the developer will work quicker.

All screens in this application are designed in this way:

1. It is defined a main screen whose name is viewport. This is the container´s views. All screen views are located in it. The viewport layout is "card" that means that an only item could be showed. To show another one, it is necessary to call a function to make the change.
2. The master view contains other two views: a search form (with buttons to realize a server request search and the other to erase data fields) and a grid with search results. As the master view has usually a "border" layout, the search form is placed either in the north or in the west, while the grid is always placed in the center. If the search form is in the north, height needs to be specified, and if it is in the west, width has to be specified too. The grid occupies the remaining space.
3. The other viewport item is the detail view. The detail of a grid selection resource in the master view is showed in this view. Also this is the view in which the user can create a new resource to be saved in the database.

Project web application views can be divided into five blocks:

- There are screens that are only composed of the view master with its search form and its grid. These are the easiest ones and their names are: Document's search and Document´s transmission. The most difficult ones that the author has found in this view, it was how to program date validation. To verify that the last date was not superior to the first date was the main

difficulty. If both dates were equals (the last hour could never be superior to the first hour), it was needed to verify the hours.

- Calendar is other simple view. As in the previous screens, this also has a grid with the years in which any user has marked public holidays and this year calendar. To create a new calendar it is enough to give a name to the unregistered year calendar and to press on the public holiday.
- Intermediate difficult screens are: Users administration, Documents format, Retentions policy, Documents type, Global parameters, Party entities administration, Community administration and Community party administration. These screens are made up of a master view and a detail view. In the master view there are a search form and a grid. This last one is replenished when an enquiry to the database has brought data. The detail view fields are fulfilled if they show the information resource selected in the grid at the master view. If fields are empty, the user can replenished them and when saved, a new resource will be created in the database.
- Similar screens to the previous ones but more difficult ones are: Installation administration, Workflows administration and Processing rules administration. In the master view there is more than one grid. These is because the resource could have relations "one to many". When the user does a search, the main gird of the view is replenished. If the user clicks on a resource, the grid directly associated is fulfilled. The detail view refers only to the selection resource in the main grid of master view. To create or to modify a secondary resource, a window is showed with the secondary resource detail. If a resource is deleted, all relation grids to the deleted resource are cleaned from the Front-End. When the request arrived to the Back-End, this deleted from the database all relation resources.
- The most difficult screen for the author is: the Dashboard. This is because: the search form changes depending on the information that the user want to watch and because it is necessary to program charts. The screen is composed of a variable search form and a "card" type layout container. A grid is associated to the container in a view. Also a chart (if it is necessary) in other view is associated. To start, the user must select a chart type: bar chart, line and point chart or without chart. Depending on values that the user selects the form fields

change. It is mandatory to submit the period of resources that the user needs to see. The period varies depending on the form fields values that the user had selected before.

The application shows different screens according to the logged user roll. When this is logged, the Back-End sends to the Front-End the screens names that he can see. The End of Degree Project is programed with the purpose that the web browser only downloads: EXT JS files and the default view. To download a view means to download: its viewcontroller, its viewmodel and the necessary models too. Because of this, the initialization time is smaller. The remaining files will be downloaded when necessary.

There is a common toolbar for all application, this is the menu. This shows all screens names that the user has access to. This is why the menu is changeable.

The project report´s author has also developed a component to control the session time. When the web client receives a response, the timeout component is renewed. If the timer expires, the inactivity user time will be checked. If it is has been interaction with the application since the last response server, a request would be sent automatically to renew the session; if not, during one minute or less, a window would be showed. This window has two buttons: one to renew the session and the other to finish it. If the window time is over, the session would finish too.

Focusing in the budget that the company provide to buy the web application, the amount of X€ is shown. This figure includes personnel expenses, hardware expenses and software expenses. It includes also the profit margin, the risk margin and the V.A.T. Although this figure is not real, the project report author has tried to make it as reliable as possible.

Web application developers have to follow client specifications. They must follow manager´s guidelines too.

8. CONCLUSION (ENGLISH)

To sum up, it will be detailed objectives and goals that the project report author has acquired during the time the project has been developed:

- The student has learnt to program in JavaScript and he has used variety methods that this language provides to program the web application.
- The author has improved his knowledge regarding HTML and CSS languages. He has understood better the communication between the client and the server thanks to the RESTfull architecture and the AJAX web development techniques.
- He has learnt to use the EXT JS 5 framework developed by Sencha. Also to search in its API and to know characteristics of its components. In addition, he has obtained an overall view of how a project is developed in a great company such as Indra.

To end up, the author would like to point out to the readers that the learning of EXT JS is complicated; but on the other side, this framework has multiple benefits such as the good documentation that the API provided, the great variety of widgets that this framework has and which were already developed by Sencha programmers (all of them are compatible ones between them), etc. Although EXT JS 5 is not qualified to develop mobile applications, Sencha Company has already lunched onto the market version 6 that it is qualified to develop them.

Finally the author wants to add that the web application buyer is quite happy with the project that the author has developed. He has been congratulated by all teammates and managers.

9. REFERENCIAS

A continuación se mostrará la lista de bibliografías que se han consultado durante la elaboración del proyecto y maquetación del Front-End de la aplicación real:

1. Estándares de la W3C para diseñar páginas web: http://www.w3.org/standards/webdesign/
2. Libro para aprender a programar en JavaScript: Douglas Crockford. "JavaScript: The Good Parts". O'Reilly Media, Inc. (2008).
3. Otro libro para aprender el lenguaje de JavaScript: David Flanagan. "JavaScript: The Definitive Guide" (6th Ed.) O'Reilly.
4. Recomendaciones de la empresa Sencha para utilizar sus productos: http://www.sencha.com/business-web-applications-why-sencha/#developers
5. Conocimientos básicos que todo programador de EXT JS debe conocer: http://www.ecured.cu/index.php/Sencha_Ext_JS
6. Documentación del framework EXT JS 5.1: https://docs.sencha.com/extjs/5.1/application_architecture/application_architecture.html
7. Documentación proporcionada por Sencha en la que se explica que es un MVC y un MVVM: http://www.sencha.com/blog/ext-js-5-mvc-mvvm-and-more/
8. Otra página que explica las diferencias entre MVC y MVVM: https://nirajrules.wordpress.com/2009/07/18/mvc-vs-mvp-vs-mvvm/
9. Ejemplos visuales de los diferentes *layout* que existen en EXT JS por defecto: http://dev.sencha.com/deploy/ext-4.0.0/examples/layout-browser/layout-browser.html
10. Libro digital en el que se explica la tecnología AJAX: https://librosweb.es/libro/ajax/capitulo_1.html
11. Concepto fundamentales sobre una arquitectura RESTful: http://asiermarques.com/2013/conceptos-sobre-apis-rest/
12. GitHub de Enrique Carrero, en el que se puede encontrar el proyecto de "HolaFifa": https://github.com/thebohodon/HolaFifa-EXTJS-5
13. GitHub de Manuel de la Vega en el que se puede encontrar varios proyectos realizados con EXT JS y otros frameworks de JavaScript: https://github.com/madelavega/KataExtJS5

Indra